Informationssystem-Entwicklung

Die integrierte Entwicklung der
Strukturierung, Funktionalität, Verteilung und Interaktivität
von großen Informationssystemen

Bernhard Thalheim

Institut für Informatik, Brandenburgische Technische Universität Cottbus
Postfach 101344, D-03013 Cottbus, Germany;
thalheim@informatik.tu-cottbus.de

Preprint BTU Cottbus, Computer Science Institute, I-15-2003
21. September 2003

Erweitertes Skriptum zu den Vorlesungen
"Datenbankmodellierung" (Teil Sprachen)[1],
"Information Systems Engineering" (Teil: Sprachen),
"Systematische Entwicklung informationsintensiver Websites" (Co-Design-Zugang)

0 Vorwort

Was diese Wissenschaft betrifft,
Es ist so schwer, den falschen Weg zu meiden,
Es liegt in ihr so viel verborgenes Gift,
Und von der Arznei ist's kaum zu unterscheiden.
Goethe Faust, Ein Fragment, Nacht, Mephistopheles

Die Spezifikation der Strukturierung, Funktionalität und Interaktivität einer Informationssystemanwendung ist Aufgabe des Informationssystementwerfers. Gewöhnlich wird eine Entwurfsmethodik empfohlen, die vom Strukturentwurf ausgeht, mit dem Entwurf der Funktionalität auf der Grundlage der entworfenen Strukturen fortsetzt und gegebenenfalls mit dem Entwurf der Oberflächen endet. Der Entwurf der Semantik kann jeweils im Anschluß an den Strukturentwurf (Entwurf der statischen Semantik) und den Funktionalitätsentwurf (Entwurf der dynamischen Semantik bzw. des Verhaltens) angeschlossen werden.

Dieser Methodik sind eine Reihe von methodischen und inhaltlichen Brüchen eigen. Die Schwierigkeit eines kompletten Datenbankentwurfs ist jedoch in vielen Fällen auf diese Brüche zurückzuführen. Es werden unterschiedliche Sprachen verwendet und unterschiedliche Personen bringen unterschiedliche Interpretation und Sichtweisen auf das Informationssystem ein.

Für die Spezifikation aller Aspekte von Informationssystemen, d.h. der Strukturierung, Funktionalität, Verteilung und Interaktivität entwickeln wir eine Reihe von miteinander integrierten **Spezifikationssprachen** . Die Sprachen unterstützen die Entwicklung von Informationssystemen mit Hilfe des Abstraktionsschichtenmodelles.

[1] Die Schrittfolge zur Entwicklung von Informationssystemen im Co-Design-Zugang wird in der Folgearbeit dargestellt. Die Komponentenkonstruktion wurde in einer Reihe von Konferenzarbeiten vorgestellt und in der Dissertation

Das Skript soll kein Theoriebuch ersetzen. Es extrahiert aus der Theorie des Entity-Relationship-Modelles [Tha00] die Elemente, die für den Entwurf großer Systeme notwendig sind. Hauptziel dieses Skriptes ist die Präsentation einer in sich geschlossenen und konsistenten Entwurfsmethode, mit der der Entwurf großer Anwendungen noch überschaubar und nachprüfbar gestaltet werden kann.

Ad-hoc-Methoden und -Sprachen bringen meist nur in einfachsten praktischen Anwendungen Erfolg. Komplexere Anwendungen sind dagegen stets eine Herausforderung für den Entwerfer. In der Literatur gibt es zwei Herangehensweisen. Entweder der Entwerfer[2] verläßt sich auf ein Entwurfstool und die damit propagierte Methodik oder der Entwerfer besitzt eine profunde Fachkenntnis, verfügt über tiefgründige Kenntnisse in der Datenbanktechnologie und ist außerdem in der Lage, beliebig abstrakt sein Wissen und seine Erkenntnisse darzustellen. Beide Zugänge haben ihre Nachteile. Ein Werkzeug, das den ersten Zugang vollständig mit trägt, gibt es noch nicht[3]. Entwerfer der zweiten Kategorie sind selten und meist nicht zur Hand.

Das Material dieses Skriptes basiert neben der Datenbanktheorie auf umfangreichen Feldstudien und auf der Kenntnis einer Vielzahl von Entwurfsprojekten, die mit dem System $(DB)^2$ (später ID^2) durchgeführt wurden[4]. Die hier propagierte Methodik wurde diesem System zugrundegelegt. Eine Vielzahl von Tips, die wir hier vorstellen, wurde aus den Projekten abgeleitet. Die hier vorgestellte Pragmatik wurde in einer Reihe von Projekten erprobt.

Die Sprachen zum Datenbankentwurf, die wir im weiteren hier umfassend darstellen, sind auch methodisch unterlegt worden. Es wurde eine Methodik mit einem schrittweisen Entwurf von Datenbankanwendungen herausgearbeitet. Diese Entwicklungsmethodik wurde sowohl in der Lehre und Projekten erprobt als auch mit den Anforderungen von SPICE 2.0 validiert. Es konnte festgestellt werden, daß unsere Entwicklungsmethodik dem SPICE Framework Stufe 3 genügt. Da nur wenige Entwicklungsmethodiken SPICE Stufe 1 erfüllen, keine andere SPICE Stufe 2 erfüllt, kann die HERM-Methodik als die modernste und ausgereifteste Methode verstanden werden.

Ich möchte meinen Mitarbeitern und Studenten in Rostock und Cottbus und meinen Projektpartnern in Chile, Deutschland, Finnland, Indien, Jordanien, Kuwait, Neuseeland, Oman, Rußland, Ungarn und Schweden sehr für die kritische Begleitung dieses Projektes danken. Mein Dank abschließend an meine Familie für ihre Geduld und ihre Inspiration.

Abkürzungen und Annotationen

Wir folgen den üblichen Notationen des ER-Modells, obwohl dies nicht einheitlich erscheint. So wird z.B. ein ER-Typ mit einer Gleichung $E = (Attributmenge, Semantik)$ eingeführt, ein Typ im allgemeinen jedoch durch seinen Konstruktionsausdruck der Form $Name < Konstruktor >< Komponentenfolge >$.

Wir verwenden Buchstaben verschiedener Alphabete mit einer Systematik:

Grundbegriffe werden mit lateinischer Schrift (z.B. T für einen Typen, f für Funktionen, $dom(B)$ für Wertebereiche), bei Kollektionen mit kalligraphischen Buchstaben bezeichnet (z.B. $\mathcal{ER}$ für ein ER-Schema)

Abgeleitete Konstrukte werden mit gotischen Schrift-Typen bezeichnet: $\mathfrak{S}$ für Sichten, $\mathfrak{C}$ für Container.

Theorien und Umgebungen werden mit erweiterten mathematischen Schriften dargestellt. Zum Beispiel bezeichnen $\mathbb{E}$ eine Gleichungstheorie und $\mathbb{S}$ eine Sitzung.

Wir unterscheiden strikt zwischen Typen, die der Spezifikation dienen, und Klassen von Objekten eines Typs. Klassen eines Typs werden mit einer hochgestellten Annotation bezeichnet, z.B. ist R^C eine Klasse von Typ R.

Anzumerken ist außerdem, daß wir hier den Notationen des ER-Modelles und der Datenbankliteratur folgen wollen. In der Informatik wird unterschieden zwischen unterlegter Theorie, Spezifikationssprache und Programm. In der Datenbankliteratur ist eine Theorie durch eine meist implizit angenommene Mengensemantik und Prädikatenlogik gegeben. Das ER-Modell ist eine Spezifikationssprache[5]. Das Datenbank-Schema ist ein Ausdruck dieser Spezifikation und ein Analogon zum Programm.

---

[2] Alle maskulinen Personen in diesem Skript sind Abstrakta. Sie gelten für feminine Personen mit.

[3] Ein Ausnahme ist das im Universitätsverbund entwickelte Werkzeug RADD sein [AAB+98].

[4] Es wurden mit $(DB)^2$ bzw. ID^2 weit mehr als 5000 große Projekte (mit mitunter mehr als 1000 Entity- und Relationship-Typen) durchgeführt, deren Ergebnisse und deren Entwurfs- und Weiterentwicklungsgeschichte dem Author vorliegen.

[5] [illegible]

1 Einführung

> Ich bin nicht ihr bester Freund.
> Ich bin ihr einziger Freund!
> *Danny DeVito* in *Das Geld anderer Leute*

Datenbank- und Informationssysteme sind heute integrierte, eingebettete oder selbständige Anwendungen und integraler Bestandteil der Infrastruktur von vielen Betrieben. Meist wird zwischen diesen Systemtypen nicht unterschieden. Wir wollen im weiteren jedoch Datenbanksysteme als die Hauptkomponente von Informationssystemen auffassen. Informationssysteme verfügen außerdem über eine Reihe von Anwendungsschnittstellen im Rahmen von Präsentationssystemen. Ein Datenbanksystem umfaßt wiederum ein Datenbank-Management-System (DBMS) und eine Reihe von Datenbanken. Information umfaßt immer eine Deutung von Daten. Information ist aus unserer Sicht nicht einfach 'Mikro'-Wissen oder eine Menge von Daten. Wir unterschieden zwischen

dem Datum als Folge von Symbolen,

Nachrichten als übermittelte Daten,

dem Wissen als validierter, wahrer Glaube bzw. zusammengefaßte, kondensierte Fakten (Daten) und Regeln und

Informationen als gedeutete Nachrichten, Daten oder Mitteilungen, die ein *Empfänger* mit bestimmten Regeln intuitiv oder explizit *auswählt* innerhalb eines *Kontextes*, *verarbeitet* und in seinen Informations-, Daten- bzw. Wissensbestand *integriert*.

Die Entwicklung eines Informationssystemes muß deshalb alle Aspekte einer Anwendung umfassen:

Strukturierung: Die Struktur eines Informationssystemes und die statischen Integritätsbedingungen werden im Datenbank-Schema zusammengefaßt, das die Strukturierung einer Datenbank beschreibt.

Funktionalität: Informationssysteme stellen eine Vielzahl von Funktionen, eine Anfrageschnittstelle, eine Modifikationsschnittstelle, eine Transaktionsverarbeitungskomponente, Programme etc. zur Verfügung. Für eine Anwendung werden Prozesse auf der Grundlage dieser Funktionen entwickelt. Für die Prozesse gelten dynamische Integritätsbedingungen. Wir fassen die Prozesse und die dynamischen Integritätsbedingungen in der Datenbank-Maschine zusammen, die die Funktionalität der Anwendung beschreibt.

Verteilung: Informationssysteme werden heutzutage in andere Systeme eingepaßt, sind selbst oft nur Bestandteile einer Infrastruktur und kooperieren miteinander. Wir entwickeln hier eine allgemeine Spezifikation der Verteilung basierend auf dem Konzept der Dienste, der Austauschrahmen und der Kooperationsbedingungen. Diese Spezifikation verallgemeinert Zugänge aus dem Bereich der Kommunikationssysteme, der verteilten Systeme und der Betriebssysteme.

Interaktivität: Ein Informationssystem soll den Benutzer bei einer Vielzahl von Aufgaben unterstützen. Es werden je nach Anwendungskontext unterschiedliche Handlungsabläufe ausgelöst. Wir fassen diese Abläufe im Story-Raum zusammen. Gruppen von Benutzern werden abstrakt durch Akteure dargestellt. Die einzelnen Arbeitsschritte fassen wir in Szenen zusammen. Die benötigte Unterstützung durch das Datenbanksystem erfolgt durch Content-Objekte, die eine Verallgemeinerung von Sichten darstellen und um eine Funktionalität erweitert wurden. Der Story-Raum und die Content-Objekte werden im Interaktionsraum zusammengefaßt.

Diese vier Aspekte müssen gemeinsam bei der Entwicklung eines Informationssystemes betrachtet werden. Wir sprechen deshalb vom integrierten Entwurf von Strukturierung, Funktionalität, Verteilung und Interaktivität eines Informationssystemes bzw. vom *integrierten Entwurf von Strukturierung und Funktionalität eines Datenbanksystemes.*

Der Entwurfsprozeß ist ein Prozeß des **Abstrahierens** und des **Konstruierens**. Wir können deshalb die unterschiedlichen Abstraktionsarten und Konstruktionsarten miteinander vergleichen.

Mit dem Zachman-Zugang [IZG97] können wir beim Konstruieren unterschiedliche Aspekte von Informationssystemen unterscheiden:

Strukturierung **(was):** Die Strukturierung der Anwendung wird durch Datenbankmodelle angegeben. Datenbanklehrbücher konzentrieren sich meist auf diesen Aspekt.

Funktionalität **(wie):** Funktionen und Prozesse, die für die Manipulation und das Retrieval benötigt werden, werden meist erst mit der Entwicklung der Funktionalität der Anwendung auf dem Niveau der Implementierung betrachtet. Da aber die Optimierung des Verhaltens der Anwendung eine dedizierte Unterstützung durch die Strukturierung erfahren muß, sollte die Spezifikation der Funktionalität und der Strukturierung abgestimmt erfolgen.

Lokalisierung **(wo):** Anwendungen sind meist verteilt auf Struktureinheiten, auf unterschiedliche Orte und auf die Infrastruktur. Die Verteilung des Datenbanksystemes war von untergeordnetem Interesse, solange eine verteilte Verarbeitung keine Effizienzvorteile brachte. Mit der Entwicklung der Vernetzung und der effektiven Unterstützung hat sich dies grundlegend geändert.

Akteure **(wer):** Mit der Entwicklung der künstlichen Intelligenz wurde auch das Mensch-Maschine-Interface komfortabler. Spezielle Schnittstellen für unterschiedliche Benutzer, je auch Fähigkeiten, Fertigkeiten, Wissen, Arbeitsaufgaben, Arbeitsumfeld, Rollen und Rechte, können mittlerweile durch DBMS unterstützt werden. Demzufolge sind die Akteure als Gruppen von Benutzern mit zu modellieren.

Zeitpunkte **(wann):** Daten altern auf unterschiedliche Art und Weise je nach der Benutzung, der Sichtweise der Benutzer, der Erneuerungsstrategie und der zur Verfügung stehenden Infrastruktur und Systeme. Der Alterungs- und Erneuerungsprozeß kann durch Modellierung der Zeitaspekte beherrscht werden.

Motivation **(warum):** Die Akzeptanz der Systeme wird stark durch die Motivation der Akteure mit bestimmt. Wir verallgemeinern die Motivationsschicht zur allgemeinen Benutzbarkeitsschicht.

Metaaspekte werden im Zachman-Modell bis auf die Motivation nicht betrachtet. Beispiele solcher Kategorien sind Qualitätskategorien wie Allgegenwart, Sicherheit, Konsistenz, Bedeutungstreue, Robustheit, Skalierbarkeit und Dauerhaftigkeit.

Benutzungsaspekte werden im Zachman-Modell vernachlässigt. Es gehören hierzu insbesondere das Aufgabenportfolio und das Organisationsmodell.

Unser Modell der Entwicklung von Informationssystemen im Co-Design-Zugang folgt den ersten drei Aspekten (Strukturierung, Funktionalität und Verteilung) und betrachtet anstelle der letzten drei Aspekte das Storyboard, d.h. die Interaktivität.

Wir fügen dem Zachman-Modell noch weitere Dimensionen hinzu:

Kompetenz **(wofür):** Es werden die Aufgaben, die durch das Informationssystem unterstützt werden sollen explizit dargestellt.

Kontext **(in welcher Umgebung):** Meist werden Kontextentscheidungen implizit in die Modellierung eingebracht. Dazu gehören nicht nur die technische und organisatorische Umgebung sondern auch die Strategie des Betreibers des Systemes.

Qualitätsgarantien **(in welcher Qualität):** Es wird explizit dargestellt, inwieweit bestimmte Qualitätskriterien durch das System unterstützt werden und welche Qualitätskriterien nicht oder nur bedingt erfüllt werden.

Laufzeitcharakteristiken **(wie derzeit):** Da die Arbeitsumgebung auch durch Ausnahmesituationen, durch aktuelle Parameter, durch zeitweilige Verschiebung der notwendigen Schritte zum Abschluß und durch benutzungsspezifische Aspekte geprägt ist, sollte die Anpassung des Systemes an die Arbeitssituation auch explizit modelliert werden.

Kollaboration (mit wem): Arbeitsaufgaben werden oft in Gruppen bewältigt. Die Kollaboration von Gruppen muß deshalb explizit dargestellt werden. Wir unterschieden zwischen Kommunikation, Kooperation und Koordination und stellen dazu Kollaborationsrahmen dar. Damit wird das Akteursmodell weiter ausspezifiziert.

Diese Dimensionen untersetzen z.T. die Zachman-Dimensionen. Da im Verlaufe des Modellierungsprozesses alle Aspekte der Anwendung explizit dargestellt werden sollten, umfaßt unsere Methodik auch diese Betrachtungswinkel.

Die Semiotik und die Linguistik unterscheiden für Sprachen drei unterschiedliche Betrachtungsweisen, die auch für unsere Spezifikationssprachen gelten:

Die **Syntaktik** (bzw. **Syntax**) untersucht die Beziehungen der Zeichen (Worte) selbst, stellt Regelsysteme zur Erzeugung korrekter Ausdrücke der Sprache bereit und führt oft zu einem Beweissystem, mit dem bestimmte Eigenschaften für Kollektionen von Ausdrücken dargestellt werden können.

Die **Semantik** untersucht die Beziehung zwischen Worten und Ausdrücken einer Sprache und den Objekten bzw. Dingen der Realität. Es werden demzufolge "Welten" Kollektionen von Ausdrücken gegenüber gestellt. Typische Gegenüberstellungen sind die Gültigkeits- bzw. die Erfüllbarkeitsrelation.

Die **Pragmatik** untersucht die Beziehung zwischen Worten und Ausdrücken einer Sprache und dem Wort- bzw. Ausdruckbenutzer und konzentriert sich auf Aspekte der Bedeutung für den Benutzer, für eine Gruppe und für einen Kontext. Die Pragmatik wird durch eine Reihe von *pragmatischen Axiomen* geprägt:

- *Man kann nicht nicht kommunizieren.* Jedes Verschweigen ist auch eine Darstellung. Im allgemeinen akzeptieren wir für die Modellierung eine closed-world-Annahme, bei der die Nichtdarstellung von Dingen der Realität auf der Irrelevanz für die Anwendung beruhen.
- *Jede Modellierung hat einen Inhalt- und einen Beziehungsaspekt, wobei der letztere den ersteren bestimmt.* Es wird implizit oder ggf. explizit die Beziehung zwischen Benutzer und System dargestellt.
- *Die Spezifikation wird durch die Interpunktion der Darstellung mitbestimmt.* Interpunktion tritt beim Austausch von Mitteilungen auf, bei der zwei Seiten eine unterschiedliche Dekomposition der Mitteilung in Bestandteile und die Bedeutungszuordnung für diese Bestandteile vornehmen. Dadurch entstehen unterschiedliche Sichtweisen auf den gleichen Ausdruck und entsprechende Beziehungskonflikte.
- *Kommunikation in den Anwendungen bedient sich digitaler Repräsentation.* Da aber die Beobachtungen oft analog möglich sind, entsteht durch falsche Digitalisierung bzw. Abtastung ggf. ein falsches Bild wie z.B. in der Monatsabrechung bei Lagerhaltungsanwendungen oder Monatsstatistiken.
- *Kommunikationsabläufe sind entweder symmetrisch oder asymmetrisch, je nachdem, ob Facetten der Kollaboration auf Gleichheit oder Unterschiedlichkeit beruhen.* Die unterschiedlichen Facetten können gleichzeitig und in unterschiedliche Symmetrierichtungen wirken und sich komplementär ergänzen wie in den Beziehungen Fachmann-Laie und Mitarbeiter-Vorgesetzter.

Neben den semiotischen Aspekten erfordert auch eine Spezifikationsmethodik eine explizite Widerspiegelung des Pragmatismus. Der Pragmatismus ist die Lehre, nach der sich das Handeln und Denken am praktischen Leben orientiert und diesem dient. Durch den Pragmatismus werden pragmatische Annahmen determiniert. Übliche pragmatische Annahmen sind die Auswahl der Sprache, die (Selbst-) Beschränkung bei der Benutzung der Sprache, die Wahl der Begriffe und ihrer Assoziationen, sowie die Wahl der Darstellungsmittel im Falle einer Auswahlmöglichkeit. Typische pragmatische und nicht dokumentierte Annahmen sind die Art der Attributdarstellung, die Auswahl der Wertebereiche und die Handlungsabläufe. Sie werden implizit vorgenommen, z.B. durch eine Annahme zur ersten

Normalform, die nur atomare Attribute zuläßt, wobei der Begriff des Atoms je nach Modellierungsurteil auch variieren kann. *Postleitzahlen* werden oft als Atom zugelassen, obwohl sie bereits aus Komponenten wie Zustellbereich und Zustellbezirk zusammengesetzt sind. Pragmatische Annahmen bilden Tatsachen, Handlungsweisen, Erfahrungen, Möglichkeiten, Potenzen und auch Fertigkeiten aus dem Anwendungsgebiet entsprechend dem praktischen Nutzen ab. Sie dienen damit dem Ziel einer möglichst effektiven Abbildung des Anwendungsgebietes.

Die Informatik hat bislang nicht allzu viele Prinzipien hervorgebracht. Die Mathematik kann man auf die Triade reduzieren: Strukturierung, Topologie und Symmetrie bzw. Erzeugung. In der Kristallographie unterscheidet man drei Grundbegriffsarten wie in Bild 1. Diese drei Prinzipien sind analog zu den Prinzipien der Quantenphysik. Dieses Modell kehrt auch in den Gesellschaftswissenschaften wieder[6]. In analoger Form kann auch die Strukturtheorie der Mathematik verstanden werden.

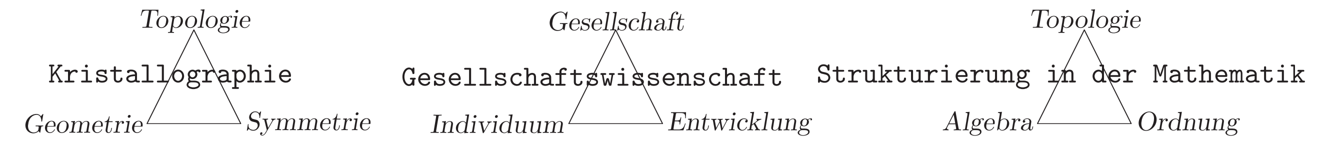


Bild 1: Die drei Prinzipien der Kristallographie, der Gesellschaftswissenschaft und der Mathematik

Die Informatik fügt diesen drei Prinzipien ein weiteres Prinzip hinzu: die Abstraktion. Das Abstraktionsprinzip ist bereits in den Ansätzen der Quantenphysik implizit enthalten und ist bei den Prinzipien der Gesellschaftswissenschaften verwirklicht. Gleichzeitig erfahren diese Prinzipien viele Ausprägungen.

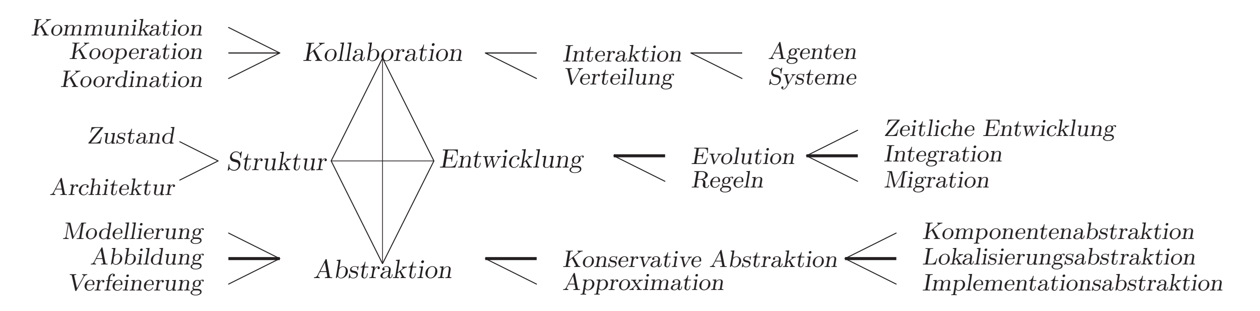


Bild 2: Die vier Prinzipien der Informatik

Jedes der vier Prinzipien besitzt unterschiedliche Facetten. So sind die Kooperation, die Kommunikation und die Koordination Facetten der Kollaboration. Eine andere Dimension von Facetten ist auch Verteilung und Interaktion. Auch für die Abstraktion können wir unterschiedliche Facetten unterscheiden: Facetten des "`wie`" (Modellierung, Abbildung, Verfeinerung) und Facetten des "`wodurch`" (Approximation, konservative Approximation).

Die Strukturierung besitzt die Zachman-Aspekte:

`womit` materialisiert: Speicher-Struktur, Repräsentationsstruktur und abstrakte Strukturen;

`wodurch` repräsentiert: direkte Darstellung und kodierte Darstellung;

`wie` konstruiert: Basis-Typen, Konstruktor-Arten und Abschlußbedingungen.

Je nach Wahl erhalten wir unterschiedliche Sprachen (bzw. "Modelle" wie das relationale oder auch objekt-relationale Modelle), Erzeugungsregeln und Materialisierungssprachen.

Diese vier Prinzipien werden in Zweigen der Informatik unterschiedlich akzentuiert. So konzentriert sich der klassische Datenbankentwurf auf die Strukturierung, verwendet eine Art der Abstraktion (die konservative Abstraktion) und integriert die Kollaboration implizit im Schema. Komponenten

[6] Diese Vorstellung haben wir leider bislang nicht in der Literatur nachweisen können, obwohl sie zur Folklore gehört.

werden innerhalb eines Schemas verschmolzen und sind dann Bestandteil einer großen Struktureinheit. Gegebenenfalls werden Aspekte der Verteilung separat behandelt. Die Entwicklung von Systemen wird dagegen gar nicht betrachtet. Da die Approximation gar keine Rolle spielt, wird sie im weiteren nicht betrachtet.

Programmiersprachen konzentrieren sich eher auf die Entwicklung von Regeln zur Zustandstransformation. Zustände werden durch eine Struktur definiert. Die abstrakten Zustandsmaschinen erlauben darüber hinaus eine Abstraktion durch Einführung einer expliziten Verfeinerungsbeziehung. Regeln können sowohl sequentiell, als auch konkurrierend als auch parallel angewandt werden. Erstmals mit den abstrakten Zustandsmaschinen wurden auch Postulate aufgestellt [Gur00]:

Postulat der sequentiellen Zeit: Zustandstransformationen erfolgen schrittweise mit einer Zeitlogik, die sequentiell ist.

Postulat der abstrakten Zustände: Zustände können durch eine Struktur über einer Signatur definiert werden, wobei Zustandstransformationen nicht die Struktur ändern und invariant gegenüber Strukturisomorphismen sind.

Postulat der beschränkten Exploration: Zustandstransformationen erfolgen für eine beschränkte bzw. endliche Menge von Zuständen des gesamten Zustandsraumes.

Oft ist es sinnvoll, die vier Prinzipien auf spezifische Art zu betrachten. In unserem Anwendungsfall betrachten wir nicht die allgemeine Kollaboration, sondern nur einige Aspekte: Kollaboration im Rahmen der Verteilung und Interaktion von System und Akteuren (anstelle von Agenten). Wir betrachten auch im wesentlichen nur die Entwicklung von Information innerhalb eines Informationssystemes und weniger die Entwicklung von Systemen selbst. Die Abstraktion wird ebenfalls nur in als konservative Abstraktion behandelt. Wir nutzen die Modellierung und konzentrieren uns weniger auf Abbildungen und Verfeinerungsmechanismen. Aus diesen vier Prinzipien leiten wir deshalb die vier Modellierungsaufgaben ab:

Modellierung der Strukturierung,
Modellierung der Funktionalität,
Modellierung der Verteilung und
Modellierung der Interaktivität.

Im Abstraktionsprozeß kann man unterschiedliche Aspekte betrachten:

- Wir unterscheiden drei `Abstraktionsarten`:
  - Die *Komponentenabstraktion* kann aufgrund unterschiedlicher Konstruktoren unterschiedliche Ausprägungen besitzen:
    - Die *Klassenabstraktion* orientiert sich auf die Unterscheidung von Instantiierung und Klassifizierung.
    - Die *Konstruktorabstraktion* orientiert sich an der Benutzung der im Datenbankmodell vorhandenen Konstruktoren. Daraus resultieren Operationen wie die Aggregation und die Dekomposition.
    - Die *Beziehungen zwischen Klassen* können explizit modelliert sein.
      - Durch *Teiltypenhierarchien* werden die Generalisierung und Spezialisierung von Klassen dargestellt.
      - Die *Konstruktionsbeziehungen* folgen meist der Definitionsbeziehung.
      - *Abbildungsbeziehungen* werden für Datenbanken auf die *Sichtenmodellierung* reduziert.
  - Die *Lokalisierungsabstraktion* orientiert auf eine Verallgemeinerung ohne Bezug zur konkreten Umgebung.
    - Die *Wiederholung von Konzepten* (Parametrisierung von Konzepten) orientiert auf der Grundlage einer Anwendungsabstraktion auf analoge Konzepte und Hierarchien artgleicher Konzepte. Der Entwurf von Einheiten kann auf verschiedene Abstraktionsebenen verteilt werden.

    - Durch *Sharing von Konzepten*, adäquate Namensgebung (Variablenkonzepte) und Verbinden kann ein Muster von Konzepten wiederholt werden.
    - Die *Wiederholung von Funktionen* kann sowohl für unterschiedliche Strukturen als auch unterschiedliche Teile der Anwendung sinnvoll sein.
    - Die *Verteilungsabstraktion* auf der Grundlage eines Namensgebungs- und Verbindungskonzeptes verbessert die Einsichtigkeit und Nachvollziehbarkeit von Konzepten.
  - Durch *Implementationsabstraktion* oder *Modularisierung von Struktur, Semantik und Operationalität* auf der Grundlage von *Verkapselung und Scoping* kann die Konzeptunabhängigkeit verbessert werden. Wichtige Methoden sind:
    - das *Verstecken von Konzepten* (Sichtenbildung) (private, Gruppen- und Weltkonzepte) und
    - *Abbildungsmechanismen für Sichten*.

- Wir unterscheiden im Informationssystementwurfsprozeß `Konstruktionsarten`. Allgemeine Hilfsmittel zur Darstellung der einzelnen abstrakten Konstrukte sind in Anlehnung an Konstruktorkonzepte die folgenden Elemente:
  - *Elementare Einheiten* zur Darstellung von Basiskonzepten,
  - *Konstruktionsregeln* zur induktiven Konstruktion von komplexeren Konzepten aus bereits konstruierten oder Basiskonzepten (die meist als Konstruktionsmethodiken verstanden werden) und
  - *Konsistenzregeln* wie Integritätsbedingungen und die 'Normalisierung' erlauben eine Sicherung der Qualitätsanforderungen.

    Einbettungsregeln ermöglichen eine Integration in den bereits vorhandenen Entwurf unter Berücksichtigung von Prioritäten, Anwendbarkeitsregeln etc.

  Zur Darstellung von Strukturierung und Funktionalität können verschiedene Repräsentationsmechanismen gewählt werden.

Darauf aufbauend sind verschiedene Entwurfsszenarios möglich:

Datenstrukturgetriebener Entwurf: Es wird zuerst die Struktur der Anwendung dargestellt, darauf aufbauend die Funktionalität und die Interaktion. Dieser Zugang wird am häufigsten im Informationssystementwurf angewandt.

Prozeßorientierter Entwurf: Es werden zuerst die Prozesse und die erwünschte Funktionalität der Anwendung dargestellt und auf dieser Grundlage die Struktur und Interaktion. Dieser Zugang wird im Rahmen der Softwaretechnologie angewandt, er ist aber für den Datenbankentwurf in dieser Ausprägung wenig sinnvoll.

Architekturdominierter Entwurf: Es wird zuerst ein "Bauplan" des Informationssystemes anhand der Anwendung abgeleitet. Die Architektur basiert auf *Komponenten* und *Assoziationen* zwischen den Komponenten. Es werden die einzelnen Komponenten unter Berücksichtigung ihrer Assoziationen und daraus entstehender Obligationen entwickelt.

Interaktionszentrierter Entwurf: Es wird zuerst der Interaktionsraum oder der Storyraum modelliert und daraus werden dann Anforderungen an die Strukturierung und Funktionalität abgeleitet. Diese Anforderungen führen zur Ableitung des Anwendungssystemes.

Weitere Strategien sind möglich, wie z.B. parallele Entwicklung verschiedener Konzepte bzw. Teilkonzepte.

Orthogonal dazu sind verschiedene Unabhängigkeitskonzepte möglich:

Unabhängigkeit des Endnutzers von spezifischer konzeptioneller Repräsentation und

Diese Unabhängigkeitskonzepte sind an der Vorgehensweise zur Implementation und der *3-Ebenen-Architektur* (Endnutzerebene, Konzeptionelle Ebene, Implementationsebene) orientiert.

In der Softwaretechnik und der Wirtschaftsinformatik wird oft eine *Herangehensweise im Rahmen eines Software-Entwicklungsprozesses* präferiert [HP97]:

1. Definition des Gestaltungsbereiches: Die bisherigen Prozesse werden rudimentär analysiert. Damit kann eine Definition der Kerngeschäftsbereiche und der wichtigsten Prozesse erfolgen.

2. Formulierung der provisorischen Ziele: Die Probleme und Schwachstellen des derzeitigen Systemes werden durch Interviews, Fragebogen, Beobachtungen und Experimente aufgefunden.

3. Analyse der bisherigen Prozesse: Die aktuell vorhandenen Prozesse werden mit entsprechenden Aktivitäten verglichen. Es wird die Systemleistung mit Meßkriterien wie Durchlaufzeit, Kosten, Fehlerquote etc. ermittelt. Die Untersuchung beruht auf einer Reihe von Qualitätsparametern:
   - Allgemeine Aspekte wie der Output des Produktes, Abnehmer, Häufigkeit der zukünftigen Änderungen,
   - Zeitaspekte wie Länge, Liegezeit, Bearbeitungszeit, Transportzeit, termingerechter Abschluß,
   - Qualitätsaspekte und Zufriedenheit wie Arbeitszufriedenheit, Anforderungen der 'Kunden', Beanstandungen, iterative Fehlerreparaturen, weitere Anpassungen des Prozesses,
   - Struktur- und Mengendaten, z.B. die Anzahl der Teilnehmer, Häufigkeit, parallele Prozesse, Rollen, Organisationseinheiten, Anzahl der Aktivitäten, parallele Aktivitäten, Adressaten, Inputinformationen, Koordinierungsaktivitäten, Verantwortlichkeit, benötigte Sachmittel,
   - Aufwand und Ertrag versus Kosten/Nutzen, z.B. Materialkosten, Informatikkosten, Personalkosten, Gemeinkosten.

4. Globale Strukturierung und Selektion eines zu verändernden Prozesses: Es wird eine Migrationsstrategie vom abzulösenden System hin zum neuen System erarbeitet.

5. Formulierung der definitiven Ziele: Es werden die Ziele an den notwendigen Verbesserungen orientiert und je nach Bedeutung für zukünftige Prozesse gruppiert und geordnet. Dadurch entsteht ein Zielportfolio mit einer Konzentration auf zentrale Ziele.

6. Ermittlung von organisatorischen Maßnahmen: Zum Erreichen des Zieles werden Maßnahmen anhand der vorher herausgearbeiteten Schwerpunktaufgaben abgeleitet.

7. Ermittlung von technischen Maßnahmen: Darauf aufbauend wird die technische Infrastruktur abgeleitet.

8. Grobmodellierung der Geschäftsprozesse: Im ersten Entwicklungsschritt wird eine Grobstruktur des zukünftigen Prozesses mit echten obligatorischen Aufgaben abgeleitet. Dazu werden Darstellungsmittel wie ereignisorientierte Prozeßketten, Information Control Nets, Process Analysis and Design Method, Petrinetze, Role Activity Diagrams, semantische Objektmodelle, Triggermodellierung genutzt. Einzelne Schritte sind dabei:
   - Modellierung des Geschäftsvorfalles,
   - Ablaufmodellierung,
   - Organisationsmodellierung nach der Iststruktur,
   - Informationsmodellierung,
   - Definition objektbezogener Business-Regeln und
   - Organisationsmodellierung nach der Sollstruktur

9. Feinmodellierung des zukünftigen Geschäftsprozesses: Es kann nun die Aufgabenverteilung für die einzelnen Partner im Entwicklungsprozeß abgeleitet werden. Diese analysieren den Daten- und Dokumentenfluß, die Entscheidungsregeln, die Geschäftsfalldaten, die Kompetenzregeln, die Kooperationsregeln, die Methodenregeln und die Zeitregeln. Auf dieser Grundlage werden einzelne Komponenten des Systemes erstellt.

10. Evaluierung der einzelnen Komponenten des Systemes: Die erstellten Komponenten werden anhand der Ziele evaluiert. Es werden außerdem Benutzungsoberflächen und die Dokumentation erstellt.

11. Systemkonfiguration: Nach Erstellung der Einzelkomponenten wird das Gesamtsystem entwickelt und konfiguriert.

12. Aus- und Weiterbildung der Mitarbeiter: Die Mitarbeiter im Betrieb werden schrittweise an das neue System herangeführt.

13. Prüfen der Systemsicherheit, Wirtschaftlichkeit und Ergonomie: Das System wird anhand von Qualitätskriterien wie

    - Sicherheitskriterien, z.B. Integrität, Verbindlichkeit, Verfügbarkeit, Vertraulichkeit
    - Wirtschaftlichkeit, wie Anpassungsfähigkeit an veränderte Prozeßabläufe, Durchlaufzeit, Durchschaubarkeit, Nachvollziehbarkeit der Prozesse, Investitionen und Betriebskosten, Zahl und Qualifikationsniveau der Mitarbeiter

    analysiert.

14. Inbetriebnahme des Systemes: Nach einem Migrationsplan wird das System schrittweise in die Praxis überführt.

Diese und andere Methodiken zeichnen sich z.T. durch sehr große Detailliertheit aus, sind aber in den wesentlichen Teilen zu unscharf und wenig brauchbar.

Ein anderer, ebenso wenig praktikabler Zugang wird in der klassischen Datenbankliteratur verfolgt. Der klassische Entwurf einer Informationssystemanwendung ist von einer Reihe von Brüchen gekennzeichnet.

Struktur-/Funktionsbruch: Die meisten Methodiken und Werkzeuge unterstützen beim Entwurf keine gleichgewichtige Sicht auf Strukturierung und Funktionalität von Informationssystemen. Prozesse werden meist nur in einer rudimentären Form spezifiziert. Durch zusätzliche Einflußnahme kann ein Administrator auch Strukturen und Funktionen im internen Schema einer Datenbank verändern. Damit kann der Zusammenhang mit dem konzeptionellen Schema vollständig zerstört werden.

Struktur-/Semantikbruch: Datenintensive Anwendungen zeichnen sich meist durch eine komplexe Struktur aus. Die statische Semantik wird entweder intuitiv durch die angewandten Konstruktoren verstanden oder erfordert, wie im relationalen Fall, tiefgründige Kenntnis der mathematischen Logik. Damit wird aber die Konsistenz in der Spezifikation entweder willkürlich oder nicht mehr nachvollziehbar.

Funktions-/Verhaltensbruch: Die Funktionen werden durch mehr oder weniger komplexe Prozesse und Operationen implementiert. Das Verhalten dieser Prozesse kann auf der Grundlage einer kompositionellen Semantik in einigen Spezialfällen hergeleitet werden. Damit ist aber nur ein Teil der dynamischen Semantik erfaßt. Sobald Prozesse zumindest in den Strukturen zyklisch werden, ist eine kompositionelle Semantik nur noch mit tiefgründigen Theorien darstellbar. Noch schwieriger ist die Darstellung der Abhängigkeiten zwischen Prozessen.

Oberflächenbruch: Verschiedene Anwender verlangen unterschiedliche Sichten auf die Datenbank und

nachträglich entwickelt, dann ist eine Vielfalt von Sichten zur Unterstützung unterschiedlicher Benutzungsarten zu entwickeln. Außerdem verlangt eine Sicht oft auch eine eigenständige Funktionalität. Diese Vielfalt ist spätestens bei einer Modifikation nicht mehr zu überschauen.

Workflow-Bruch: Geschäftsprozesse können analog zu langandauernden Transaktionen im Ablauf unterbrochen werden, auf anderen Geschäftsprozessen basieren und unterschiedliche Granularität besitzen. Damit entsteht ein komplexes Ausführungsmodell, das von einem Normalentwickler nicht mehr überschaut wird.

CASE-Tool-Bruch: Die meisten Entwicklungsumgebungen erlauben, wenn sie über reine Malprogramme hinausgehen, nur eine Einbahnstraße in der Entwicklung. Nach der Erzeugung des logischen Modelles aus dem Entwurfsmodell ist es in der Regel unmöglich oder zumindest sehr schwer, beide Modelle miteinander konsistent zu halten. Es ist deshalb eine 'harte Kopplung' der konzeptionellen, externen und internen Modelle erforderlich. Jede Modifikation eines Schemas zieht ansonsten schwierige Reorganisationen der Datenbank nach sich.

Diese Brüche entstehen durch unterschiedliche Ziele im Verlaufe des Entwicklungsprozesses, wie z.B.

- Konzentration auf einen Aspekt ohne Berücksichtigung anderer Aspekte oder
- Verfügbarkeit einer bestimmten (zumeist unvollständigen) Entwicklungsumgebung oder einer bestimmten Entwicklungsmethodik

und resultieren in

- *unterschiedlichen Spezifikationssprachen* und
- *unterschiedlicher Semantik und Bedeutung der einzelnen Sprachkonstrukte.*

Außerdem implizieren sie eine *Nichtberücksichtigung der Bedürfnisse des Endbenutzers*.

Im Datenbankentwurf wird die Struktur, Funktionalität und Semantik einer Datenbankanwendung so spezifiziert, daß die infragekommende Anwendung auf einer Plattform bzw. einem Datenbankverwaltungssystem (DBMS) effizient verwaltet und bearbeitet sowie in benutzerfreundlicher Form dargestellt werden kann. Damit ist neben der *Speicherkomplexität* und der *Verarbeitungskomplexität* auch die *Einfachheit der Benutzung* zu optimieren. Diese Aufgabe ist sehr komplex. Aufgrund dieser Komplexität tendieren viele Entwurfsmethodiken zu einem **Teilentwurf** oder einem **partiellen Entwurf**. Oft wird nur die Struktur einer Anwendung entworfen. Die Semantik wird z.T. als intuitiv erklärt vorausgesetzt. Es wird dann angenommen, daß - auf der Grundlage der aus der Struktur ableitbaren generischen Operationen und Transaktionen - jede Funktion auch in einfacher Form dargestellt und entwickelt werden kann. Da 4GL-Sprachen eine benutzerfreundliche Notation nachgesagt und deshalb eine Benutzeroberfläche nicht entwickelt wird, ist ein Datenbanksystem nach wie vor extrem benutzerunfreundlich. Viele DBMS erlauben deshalb die Erstellung von sichtenbasierten Formularen bzw. Menüs. Aufgrund dieser Vorgehensweise wird durch die Struktur einer Anwendung die gesamte Funktionalität und Benutzbarkeit durch den Strukturentwurf dominiert.

Der Datenbankentwurf ist Bestandteil jedes Datenbankkurses. Zwischen 30 und 50 % des Umfanges von Datenbankbüchern werden diesem Teil gewidmet. Oft wird z.B. in der folgenden Reihenfolge vorgegangen: Struktur des Entwurfsprozesses, Anforderungsanalyse, Modellierung mit dem Entity-Relationship-Modell, relationale Modellierung und Normalisierung, objekt-orientierte Modellierung, Sichtenentwurf, Übersetzung in logische Datenmodelle, physischer Entwurf, verteilte Datenbanken, Tuning und Optimierung. Eine Methodologie für den Datenbankentwurf ist damit jedoch nicht gegeben. Eine Methodik[7] wird allerdings durch die Reihenfolge der Kapitel vorgegeben.

Diese oft empfohlene, aber den Entwerfer grausam überfordernde Methodik bedeutet, für jeden Schritt ein anderes Modell zu verwenden: für die Anforderungsanalyse ein Fragment der natürlichen Sprache, für den Strukturentwurf das Entity-Relationship-Modell, für den Semantikentwurf das

[7] Eine Methodik, die auf der strukturellen Rekursion aufsetzt, besteht i.a. aus drei Bestandteilen: einer Sprache zur Darstellung der Urteile (Entwurfsurteile), einer Menge von Regeln zur Konstruktion neuer Urteile und einer Menge von Konsistenzregeln, mit denen falsche Konstruktionen ausgesondert werden können. Eine Entwurfsentscheidung geht

relationale Modell, für den operationalen Entwurf eine Methodik der Softwaretechnologie, für den physischen Entwurf verschiedene Datenstrukturen, für das Tuning ein operationales Modell etc. Die Beschränkung ist nicht nur, daß kaum jemand alle diese Modelle im Detail beherrscht und filigran anwenden kann, sondern das damit verbundene Abbildungs- und Konsistenzproblem. Man entwirft mit einem Modell, setzt diesen Entwurf im anderen Modell fort und muß die bisherigen Resultate in das andere Modell transformieren. Dabei geht meist bereits entwickeltes Entwurfswissen verloren und muß neu entwickelt werden. Hier verwenden wir dagegen durchgehend ein erweitertes Entity-Relationship-Modell, das es gestattet, das vollständige Entwurfswissen in nur einem Modell darzustellen. Die Transformation auf die logischen und physischen Modelle ist bereits seit längerer Zeit vollständig erforscht. Diese Transformation kann uns ein Entwurfswerkzeug vollständig abnehmen.

Wenige geübte Datenbankentwerfer sind in der Lage, beim Strukturentwurf auch die Funktionalität, die Benutzbarkeit und die Effizienz in Einklang zu bringen. Diese 'Genialität' wird jedoch nur in jahrelangem Training erworben und ist spätestens bei einer Modifikation der Anwendung, die bereits meist nach kurzer Einführungszeit erfolgt, zum Scheitern verurteilt. Deshalb benötigen wir eine *Entwurfsmethodik, die Struktur, Funktionalität, Benutzbarkeit und Effizienz in gleichem Maße berücksichtigt*.

Die Qualität von Schemata wird bestimmt durch:

1. Für den Benutzer:

   Natürlichkeit impliziert ein einfaches Verstehen und einfaches Formulieren von Anfragen. Deshalb ist für die Akquisition die Darstellung `semantischer Einheiten` von zentralem Interesse. Schemata werden leicht lesbar und selbsterklärend, wobei enkryptische Namen vermieden werden und die Bedeutung einfach erhalten werden kann. Dadurch werden Integritätsbedingungen in verständlicher Form formulierbar und künstliche abstrakte Typen vermieden.

   Minimalität impliziert ein eindeutiges Verstehen der Komponenten des Schemas. Unterschiedliche Gesichtspunkte werden vermieden. Ein Schema ist konzeptionell minimal, wenn nicht alle möglichen Teilfälle, sondern nur die relevanten dargestellt werden.

   Sichtendarstellung für einzelne Benutzergruppen unterstützt die Verständlichkeit und die Benutzbarkeit des Schemas.

   Systemunabhängigkeit und das Ausschließen unnatürlicher Systembeschränkungen ermöglichen eine Konzentration auf die inhaltlichen Konzepte der Anwendung.

   Verständliche Darstellung komplexer Zusammenhänge vereinfacht das Erfassen und Verstehen komplexer Integritätsbedingungen und eine hohe Anzahl von Integritätsbedingungen.

   Ein Verständnis der Speicherung gibt dem Benutzer einen intuitiven Überblick über die Implementation der Datenbank.

2. Für die Unterstützung durch das System:

   Wenig oder keine Redundanz verringert den Pflegeaufwand, der durch das System zu leisten ist. Damit werden Inkonsistenz und update-Anomalien vermieden. Mitunter ist eine Pflege so aufwendig, daß kein System diese leisten kann.

   Durch Systemunabhängigkeit wird eine Portierbarkeit erleichtert.

   Durch eine adäquate Sichtenunterstützung kann jede Sicht eines Benutzers auf einfache Weise unterstützt werden.

3. Für die Anwendung:

   Bei Vollständigkeit werden alle Aspekte der Anwendung, die notwendig sind, repräsentiert.

   Durch Flexibilität bedingen Änderungen in der Anwendung nicht sofort Änderungen aller Teilschemata.

   Werden relevante Dinge repräsentiert und nicht alle möglichen Situationen, dann kann ein Schema einfacher gepflegt, erweitert und verstanden werden.

Betriebliche Modelle dienen der Repräsentation betrieblicher Einschränkungen (operationale Beschränkungen, Gesetze, Regulierungen, Planung, Kontrolle, etc.).

Daraus können Prinzipien des Entwurfes abgeleitet werden:

- Was wird modelliert? In einer korrekten Repräsentation verkörpert jeder dargestellte Typ Objekte einer bestimmten Klasse in der realen Welt und jede relevante Klasse wird aufgezeigt. Der Grad der Detailliertheit wird nur soweit vorangetrieben, daß Anfragen und Updates in einer einfachen Form möglich sind, aber zugleich soweit, daß die Entwicklung von Anwendungen unterstützt wird. Prinzipiell werden nur stabile Strukturen repräsentiert. Teiltypenhierarchien können ansonsten bis ins letzte Detail aufgespleißt werden, so daß jede Änderung in der Anwendung eine andere Hierarchie bringt.
- Die Darstellung semantisch sinnvoller Einheiten ist so einfach wie möglich zu gestalten. Damit ist die Bedeutung einfach herauslesbar. Jede semantische Einheit besitzt eine einfach erklärbare Bedeutung.
- Jeder Fakt wird nur einmal repräsentiert, wodurch Anomalien vermieden werden. Jede Assoziation erscheint nur einmal. Zerlegbare Fakten sollten in Abhängigkeit von den updates auch zerlegt dargestellt werden. Beispiele eines ungünstigen Entwurfes sind solche, die eine update Anomalie besitzen. Surrogat-Attribute werden demzufolge erst auf logischen Niveau wirksam.
- Durch Sicherung der Identifizierbarkeit jeden Faktes wird auch eine Modifikation einzelner Fakten ermöglicht.
- Durch eine saubere Unterscheidung der Nullwerte (unbekannt, nicht anwendbar, etc.) kann auch eine entsprechende Funktionalität unterstützt werden. Nicht anwendbare Werte deuten auf unsaubere Modellierung. Eine bessere Modellierung ist die Darstellung durch Teiltypen. Schwierigkeiten bei Anfrageauswertung und -formulierung können so umgangen werden. Es gibt strukturelle Nullwerte und Ausnahmenullwerte.
- Wir benötigen klare Regeln für die Zuordnung zu den Konzepten (Attribut oder Entity-Typ oder Relationship-Typ). Mitunter muß auch für Konzepte, die eigentlich durch Attribute dargestellt werden, ein Entitytyp eingeführt werden.
- Attributnamen dienen einer intuitiv verständlichen Charakterisierung von Objekten der Datenbank.
- Hierarchische Strukturen sind meist einfacher zu behandeln. Insbesondere wird die Pflege der Integritätsbedingungen und die Generierung von Operationen einfacher.
- Surrogate sollten im konzeptionellen Entwurf nur in Ausnahmefällen verwendet werden. Modifikationen werden ansonsten schwieriger.

Damit können kritische Faktoren für die Entwicklung einer Entwurfsstrategie abgeleitet werden:

1. Ein schrittweiser Entwurf kann unterstützt werden.
2. Rollen und Verantwortlichkeiten müssen wohldefiniert sein.
3. Eine klare Unterscheidung zwischen allgemeinen und produktspezifischen Entwurfstechniken erleichtert die Migration zu anderen Werkzeugen.
4. Das Datenwörterbuch (data dictionary) sollte auch Versionen und weitergehende Informationen enthalten.
5. Der Entwurf basiert auf einem und nur einem Modell, das mindestens die gesamte Funktionalität

6. Durch die Darstellung der Entwurfsentscheidungen für ein späteres Reviewing und Einführung von Checkpoints, denen sich Entwerfer unterwerfen müssen, insbesondere zum Einholen von Kompetenz, kann eine spätere Modifikation und die Diskussion von Varianten vereinfacht werden.

7. Der Struktur-, Funktions- und Semantikentwurf wird integriert durchgeführt.

8. Durch übersichtliche Repräsentationstechniken wird ein Entwurf intuitiv auch in seiner Entwurfsgeschichte verständlich.

Außerdem muß eine entsprechende Transformationstechnik existieren, mit der ein Prototyping, z.B. in relationalen DBMS, erleichtert wird.

In diesem Skript wird eine Methodik vorgestellt, die sich ein Entwerfer selbst verändern kann. Wir gehen davon aus, daß jeder Entwerfer seine eigene Methodik verwendet. Es gibt zwar Gemeinsamkeiten, aber die Wahl der Methodik hängt nicht nur von den Kenntnissen und Erfahrungen des Entwerfers ab, sondern wird auch durch das Anwendungsgebiet und durch die Projektpartner mitbestimmt. Deshalb wird im Skript auch dargestellt, wie man die Methodik, die im Hauptteil des Co-Design-Buches vorgestellt wird, durch eine eigene Methodik ersetzen kann. Unsere Methodik hat sich in den mehr als 100 $(DB)^2$-Anwendergruppen als eine der am häufigsten und am weit verbreiteten Methodiken erwiesen. Neben dieser Methodik existieren viele verschiedene andere Methodiken.

Die Modellierung wird immer von der Verfeinerung begleitet. Verifikation und Validierung dienen der Kontrolle der Resultate wie in Bild 3 dargestellt.

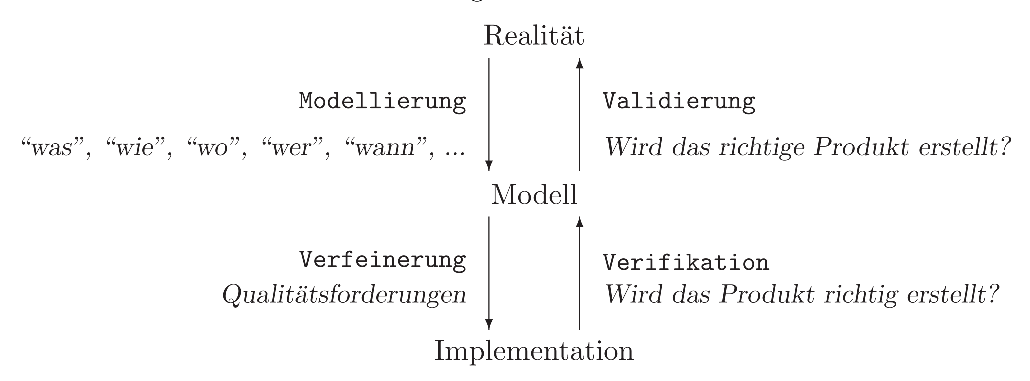


Bild 3: Modellierung, Verfeinerung, Verifikation und Validierung

Obwohl ein Datenbankentwurf immer für eine bestimmte Umgebung und damit für eine bestimmte Plattform durchführt wird, sollte der Entwurf zuerst die Anwendung adäquat widerspiegeln und zuletzt erst durch die Implementationseinschränkungen der gewählten Plattform getragen werden. Ein solcher Entwurfszugang ist erst durch die Entwicklungen der Datenbanktechnologie und der -theorie während der letzten 10 Jahre ermöglicht worden. Es gibt erst in Ansätzen methodische Umsetzungen auf dem internationalen Markt.

In diesem Skript stellen wir eine Zugang vor, der auf tiefliegenden theoretischen Erkenntnissen beruht. Es ist in diesem Skript nicht unser Ziel, die Datenbanktheorie in aller Tiefgründigkeit vorzustellen, sondern eine Methodik zu entwickeln, die auf den Erkenntnissen dieser Theorie beruht, diese aber nicht vordergründig verwendet. Viele der Tips in diesem Skript haben Lehrsätze im Hintergrund. Wir versuchen weiterhin, die Fallen und Untiefen, die mit ungeschickten Methodiken verbunden sind, zu vermeiden oder zu umschiffen. Dadurch wird auch eine Reihenfolge der Entwurfsschritte mit diktiert.

Es gibt eine umfangreiche Literatur zum Datenbankentwurf auf der Grundlage des relationalen Modelles. Das relationale Modell eignet sich jedoch nur für einige Entwurfsphasen. Die Semantik und der Zusammenhang zwischen den relationalen Schemata ist nur relativ umständlich und abstrakt darstellbar. Das damit erforderliche Abstraktionsniveau überfordert auch aufgrund der Komplexität die Entwerfer. Selbst die 'Perle' der relationalen Theorie, die Normalisierungstheorie, erfordert vom

Entwerfer umfangreiche und tiefgehende Kenntnisse. Die Werkzeuge generieren meist nur eine von vielen möglichen Normalisierungen, so daß eine Korrektur per Hand oft erforderlich ist. Aus diesem Grund hat sich das auf eine graphische Darstellung stützende Entity-Relationship-Modell für die ersten Entwurfsphasen durchgesetzt. Es gibt heute fast kein Entwurfssystem, das dieses Modell nicht in irgendeiner Form benutzt. Wir folgen diesem Trend, erweitern aber das Modell, um auch die anderen Entwurfsphasen mit diesem Modell durchführen zu können.

Es gibt allerdings bislang keine Theorie der Modellierung von Informationssystemen[8]. In der Literatur finden sich nur einige Bestandteile einer solchen Theorie: Theorie relationaler Schemata, Theorie der Petri-Netze, Theorie von Workflows. Wir benötigen einen vollständigen Zugang, der eine Modellierung der Strukturierung, Funktionalität und Interaktivität unterstützt. Außerdem sollten Aspekte der Verteilung dargestellt werden. Unsere Theorie stützt sich auf zwei Darstellungssprachen: das erweiterte Entity-Relationship-Modell (HERM) und die Webseiten-Beschreibungssprache SiteLang, sowie der Verteilungssprache DistrLang. Mit der ersteren können wir alle datenbank-basierten Aspekte wie Strukturierung, Funktionalität, Verteilung und Sichten-Suiten (als Verallgemeinerung des Sichtenbegriffes) darstellen. Mit SiteLang können wir alle Aspekte der Interaktivität und der Einbettung von Datenbanksystemen in interaktive Systeme darstellen. SiteLang umfaßt neben der Darstellung von Interaktion und Stories auch die entsprechenden Kontextbedingungen, zu denen insbesondere der Gestaltungsrahmen, der Kommunikationsrahmen und der Arbeitsrahmen gehören. DistrLang stellt die Dienste und die Kollaboration für die Verteilung dar. Die unterschiedlichen Elemente unseres Modellierungsansatzes sind auf Seite 18 zusammengefaßt.

Modellieren ist das Herstellen, Modifizieren, Analysieren und Nutzen von Modellen zur Herstellung von Vorstellungen zu Dingen der Realität. Der Modellbegriff basiert auf drei Abstraktionsmerkmalen:

- Abbildungsmerkmal: Ein Modell stellt einen Ausschnitt der Realität dar. Es werden somit Objekte Dingen der Realität zugeordnet.
- Verkürzungsmerkmal: Ein Modell abstrahiert von den Eigenschaften der Realität. Es werden nur einige “relevante” bzw. “wichtige” Eigenschaften dargestellt.
- Pragmatisches Merkmal: Ein Modell wird nicht ein für alle mal bestimmt, sondern hängt von den Zeitpunkten, dem Anwendungskontext und den Auffassungen der beteiligten Individuen ab. Diese können sich jederzeit ändern.

Damit werden in der Modellierung *Urteile* durch den Modellierer gefällt, welche Dinge der Realität für welchen Ausschnitt mit welchen Eigenschaften von Interesse sind. Es gibt eine ganze Reihe von Urteilen, die für uns von Interesse sind:

- *Existenzurteile* konstatieren die Existenz von Dingen.
- *Belegurteile* dienen dem Belegen von Beobachtungen.
- *Beziehungsurteile* stellen Dinge in ihren Beziehungen dar.
- *Bestimmungsurteile* dienen der Assoziierung von Urteilen mit Eigenschaften.
- *Assoziierungsurteile* erlauben die Assoziierung, die Aggregation und die Komposition.
- *Abhängigkeitsurteile* stellen semantische Beschränkungen dar.

Ein Urteil ist bei weitem nicht absolut. Wir stellen deshalb die Modalität explizit dar. Die Modalität erlaubt je nach Urteilsart auch die Entwicklung logischer Theorien. Ein Modellierungsurteil kann eine *Annahme*, eine *Meinung*, eine *Hypothese*, eine *Gedankenverbindung* oder auch eine *Frage* darstellen.

Ein Modellierer ist ein Individuum, das in einem Kontext (z. B. der Anwendung oder in einem kulturellen Kontext) Urteile fällt. Oft folgt das Modellierungsurteil einer Referenzdarstellung. Demzufolge fassen wir ein Modellierungsurteil als ternäre Beziehung zwischen Eigenschaft, Theorien und

[8] Einen ersten Ansatz liefert die Arbeit [Kas03], in der ein Theorieansatz angegeben wird. Wir verdichten diesen

den im Kontext agierenden Individuen auf. Weiterhin kann im Entwicklungsprozeß ein Urteil wieder revidiert werden.

Unsere Vorstellung vom Modellierungsurteil haben wir in Bild 4 in vereinfachter Form zusammengefaßt.

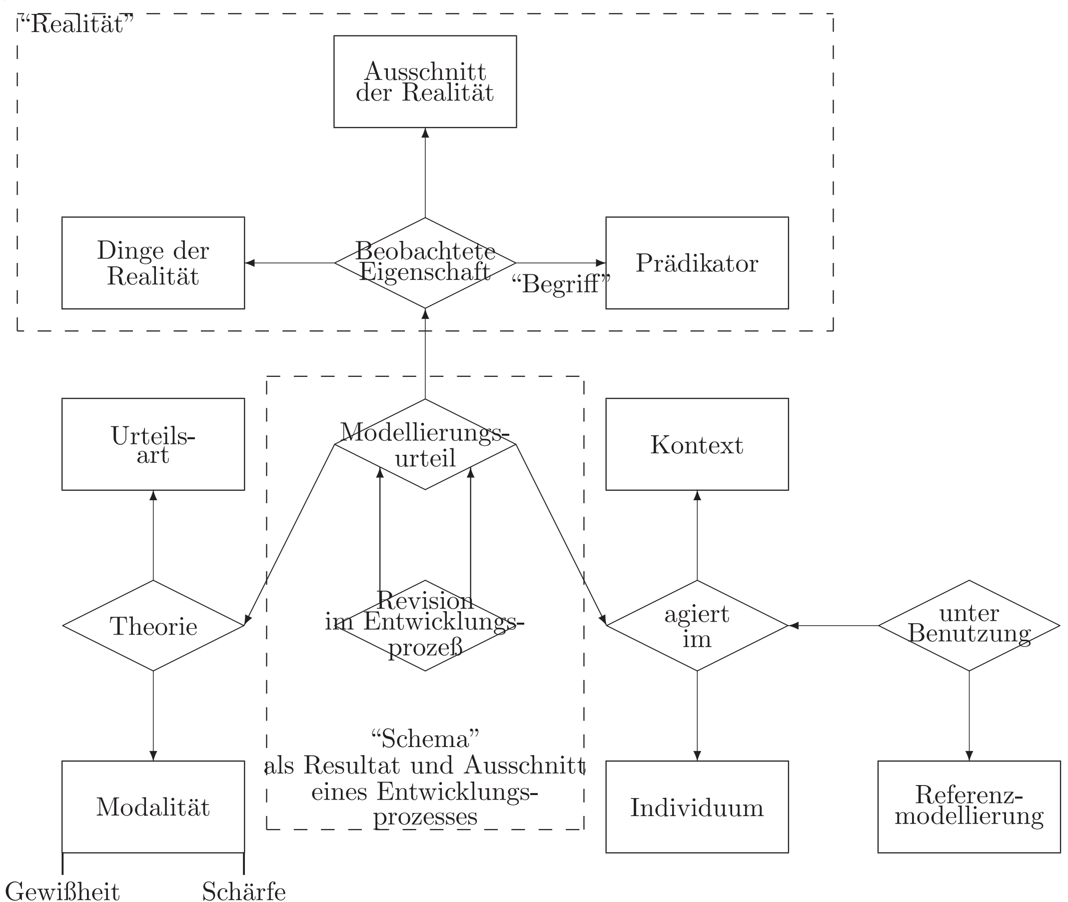


Bild 4: Modellierungsurteile durch Individuen im Modellierungskontext und das Dilemma der Modellierung

Mit der Darstellung in Bild 4 wird gleichzeitig auch das Dilemma der Modellierung sichtbar. Sind nach der Modellierung nur noch die Modellierungsurteile verfügbar, dann sind nicht mehr die impliziten Annahmen, die theoretischen Grundlagen, die Beobachtung der Realität und oft auch die Spezifika des Entwicklers nachvollziehbar. Damit entstehen Schemata, die der Nachwelt nicht mehr verständlich erscheinen, die zu einer Doppelentwicklung innerhalb von großen Anwendungen, wie z.B. bei SAP R/3, führen und neben Redundanz- auch erhebliche Konsistenzprobleme besitzen.

Redundanz kann eine sinnvolle Eigenschaft sein, sollte aber explizit erfaßt und gepflegt werden. Inkonsistenz ist selten sinnvoll. Vollständigkeit ist eines der Hauptkriterien bei der Beurteilung der Qualität neben diesen Kriterien.

Ein wichtiger Problemkreis vor Einführung eines Informationssystemes ist das Abwägen, ob dieser Einsatz nicht zu höheren Kosten führt. Der Nutzen von Informationssystemen besteht

1. in der gemeinsamen und parallelen Benutzung von Daten bei gleichzeitiger Benutzung unterschiedlicher Sichtweisen auf die Daten,

2. in kontrollierter Redundanz von gleichen Datenbeständen,

3. in der Unterstützung von eingeschränktem Benutzerzugriff, die auch Sicherheitsmechanismen einschließt,

4. in der Bereitstellung verschiedener Schnittstellen für unterschiedliche Benutzungsgruppen,

5. in der Darstellung komplizierter Beziehungen der Daten,

6. in der Bereitstellung von Mechanismen zur automatischen Integritätserzwingung und
7. in einer Robustheit, die einen Wiederanlauf auch nach Systemfehlern erlaubt.

Ein Einsatz von Datenbank-Management-Systemen (DBMS) ist besonders sinnvoll

- zur Unterstützung heterogener Benutzergruppen, die eine gemeinsame Datenhaltung präferieren,
- falls keine oder kontrollierte Redundanz gewünscht ist,
- falls eine Benutzerverwaltung und Authorisierung sinnvoll ist,
- falls unterschiedliche Schnittstellen für unterschiedlich geschulte Benutzer bereitgestellt werden sollen,
- falls komplexe Daten oder komplexe Beziehungen zwischen den Daten vorliegen,
- falls Integritätsmechanismen genutzt werden sollen,
- falls eine Fehlerbehandlung und Archivierung erforderlich ist und
- falls eine geringe Entwicklungszeit für sich ändernde Anwendungen bevorzugt wird.

Ein DBMS sollte man nicht benutzen,

- wenn ein hoher zusätzlicher Aufwand entsteht
  - durch hohen initialen Aufwand für Hardware und Software bei geringem Nutzen durch den späteren Betrieb,
  - durch hohe Allgemeinheit der vorhandenen Funktionen und
  - durch die Einführung von Algorithmen zur Unterstützung von Sicherheit, konkurrierenden bzw. parallelen Betrieb,
- wenn die Anwendung und die Datenbank eher einfach sind,
- wenn Real-Zeit-Forderungen nicht vom DBMS unterstützt werden können und
- wenn kein Mehrfachparallelzugriff auf Daten vorliegt.

Das Skript beabsichtigt nicht, eine vollständige Einführung in die Datenbank- oder zumindest in die Datenbankentwurfsliteratur zu geben. Das Literaturverzeichnis wurde bewußt kurz gehalten[9]. Die Referenzen in [Tha00] und [Tha91], sowie in [GMUW00] und [FvH89] sind stattdessen für weitere Studien zu verwenden. Wir gehen in diesem Skript davon aus, daß der Leser bereits grundlegende Begriffe der Datenbanktechnologie kennt. Eine Reihe von Fachbegriffen, die standardisiert verwendet werden, werden deshalb nicht nochmals eingeführt[10].

Dieses Skript konzentriert sich auf die Spezifikation der konzeptionellen, Benutzer-, Geschäftprozeß- und strategischen Schicht. Deshalb werden Aspekte der Darstellung auf logischer oder physischer Schicht vollständig ausgelassen. Für die Spezifikation von Strukturierung und Funktionalität auf logischer Sicht verweisen wir auf [Tha03]. Wir wollen kein XML- oder auch HTML-Buch ersetzen. Dieser Buchmarkt ist unübersichtlich und strotzt vor vorgespiegelter Einfachheit. Unter den soliden Einführungen sticht [KM03] hervor. Zum Storyboarding gibt es leider auch meist nur Erzählliteratur von Autoren, denen eine sehr kleine Website als Illustration und Erfahrungshintergrund dient. Zur Spezifikationstheorie verteilter Systeme auf logischer Sicht kann am besten [ALSS03, DGH03] herangezogen werden. Auf höherem Abstraktionsniveau existiert unserer Beobachtung nach keine einzige Literaturquelle.

[9] Die Bibliographie in [Tha00] ist bereits länger als 50 Seiten.

[10] [illegible]

	Zugrundegelegte Theorie	Verwendete Sprache	Instanzenwelt		Modell zur itera	
			Klasse	Objekte	Schema	Aggregation
Relationales Paradigma	Mengenlehre	Relationenalgebra bzw. -kalkül	Tabelle	Tupel	relationales Schema	implizit durch Integritätsbedingungen
ER-basierte Struktur	Mengenlehre	HERM	Klasse	Objekte	ER-Schema	Relationship-Typ
Statische ER-Semantik	Prädikatenlogik	HERM	-	-	Semantik des ER-Schemas	Statisch
ER-basierte Prozesse	Abstract State Machines	HERM	Workflow-Klasse	Workflow-Objekt	Workflow-Schema	Programm
Dynamische ER-Semantik	Temporale Logik	HERM	-	-	Semantik des Workflow-Schemas	Dynamis
Unterstützende Sichten-Suite	HERM-Algebra	HERM	Sichtenklasse	Sichtenobjekt	Sichtenschema	Sichtentyp
Interaktion	Mengenlehre	SiteLang	Content-Klasse	Content-Objekt	Content-Typen-Raum	Content-Typ
Stories	(Story-Boarding)	SiteLang	Szenario	Szenarium	Storyraum	Story
Modellierung von Benutzern	Organisationstheorie	SiteLang	Klasse von Benutzern	Benutzer	Akteurschema	Gruppen
Kollaborationsrahmen	*Gruppenarbeit*	*SiteLang*	*Kollaborationsvertrag*	*Arbeitsgruppe*	*Kollaborationsschema*	*Zusammenwirken*
Gestaltungsrahmen	*User-Interfaces*	*SiteLang*	*Interfaceklasse*	*Interface*	*Arbeitsplatzsuite*	*Arbeitsplatz*
Arbeitsrahmen	*Methoden des Problemlösens*		*Auftragsklasse*	*Auftrag*	*Arbeitsfeld*	*Portfolio*
Verteilung	Abstract State Machines	DistLang	Akte, Logfile	Protokollinstanz, Kontraktinstanz	Schema des verteilten Systems	Komponente, Architektur

Modellierung der Strukturierung: ER-basierte Struktur und statische ER-Integritätsbedingungen

Modellierung der Funktionalität: ER-basierte Prozesse und dynamische ER-Integritätsbedingungen

Modellierung der Verteilung: Schema des verteilten Systemes mit Architektur und Komponenten, sowie deren

Modellierung der Interaktivität: Content-Typen und Story-Raum mit Akteuren unterlegt durch Arbeits-, Kor

Sichten-Suiten zur Unterstützung der Interaktivität und Verteilung: Sichtenschema mit Sichtentypen

2 Sprachen zur Modellierung von Informationssystemen

Denn eben, wo Begriffe fehlen,
Da stellt ein Wort zur rechten Zeit sich ein.
Mit Worten läßt sich trefflich streiten,
Mit Worten ein System bereiten,
An Worte läßt sich trefflich glauben,
von einem Wort läßt sich kein Jota rauben.
J.W. Goethe, Faust, Erster Teil, Studierzimmer, Mephistopheles

Wir wollen im weiteren zeigen, wie sich die vier Aspekte Strukturierung, Funktionalität, Interaktivität und Verteilung verbinden lassen. Eine allgemeine Vorstellung der integrierten Elemente vermittelt Bild 5.

Generelle Aspekte von Informationssystemen

Strukturierung
Funktionalität
Interaktivität
Verteilung
Qualität
Struktur
Statische Integritätsbedingungen
Prozesse
Dynamische Integritätsbedingungen
Content-Objekte
Stories
Dienste
Kollaborationsrahmen
Szenarien
Aufgaben
Benutzergruppen

Konzeptionelle Spezifikation von Informationssystemen

ER-Schema
Datenbank-Maschine
Interaktionsraum
Diensteraum
Content-Typen
Story-Raum Drehbuch
Architektur
Sicht
Funktionalität
Container
Szenen
Akteure
Kollaborationsrahmen

Logische Spezifikation von Informationssystemen

relationales Schema
XML-Schema
... Schema
Stored procedures
Transaktionen
Programme, Trigger
Dialogverwaltungssystem
Oberflächen
Dienste- und Kollaborationsverwaltungssystem
Verteilung, Protokolle, Qualität

Bild 5: Integrierte Entwicklung von Strukturierung, Funktionalität, Interaktivität und Verteilung

Wir beabsichtigen, den vollen Entwicklungsprozeß in seiner *Gesamtheit* zu begleiten. Deshalb ist

für eine Spezifikation eines Informationssystemes auch eine Beschreibung der Datenbank-Maschine und eine Beschreibung der Content-Objekte und des Story-Raumes sowie des Diensteraumes notwendig.

Die konzeptionelle Beschreibung umfaßt auch der Beschreibung der Funktionsweise eines implementierten Informationssystemes, d.h. auch die Beschreibung der Content-Typen und des Story-Raumes. Gewöhnlich wird dieser Teil dem Präsentationssystem zugeordnet und erst später entwickelt. Damit werden Performanzengpässe von Anfang an mit ausgelöst.

Wir führen hier erstmals eine allgemeine Theorie der **Content-Objekte**[11] und Content-Typen ein. Content ist ein derzeit häufig überladener Begriff. Man verlangt heute von einem Content-Management-System (CMS), daß folgende Teilsysteme und Lösungen eingeschlossen werden.

- Portal-Verwaltung, Enterprise Content Management;
- Content-Anlieferung, Agentur-Lösungen, Content-Provider, Customer Relationship Management, E-Commerce-Lösungen, E-Marketing, Online Payment;
- Dokument-Verwaltung, -Archivierung, und -Suche, Unterstützung von Dokumenten-Arbeitsabläufen;
- Intelligente, benutzerspezifische Erzeugung von Inhalten;
- ASP-Lösungen, Media Asset Management;
- Group-Ware-Lösungen, Intranet-Lösungen;
- Redaktionssystem, Ausspielsystem, Erneuerungssystem;
- Skalierbare Lösung, Agententechnologien, Performance Monitoring, Sicherheitstechnologien, Hochverfügbarkeit;
- Open-Source-Lösungen, Community-Lösungen.

Diese Liste ist zu umfangreich. Außerdem wird damit der Begriff Content vollständig überladen. Stattdessen bevorzugen wir eine *Separation von Gesichtspunkten, Begriffsbildungen und Aspekten* so, wie sich dies in der Semiotik, der Linguistik und der Mathematischen Logik eingebürgert hat. Wir unterscheiden deshalb zwischen

Content als Extension eines Referenzobjektes (Intension), als eine Menge oder i.a. Kollektion von Daten, Nachrichten oder Informationen,

Konzept als Plan, als Zusammenfassung eines Vorhabens oder Begriffes in konsistenter, überschaubarer und nachvollziehbarer Form, mit dem die Gesamtheit der Merkmale zusammengefaßt wird und

Begriff als Intension oder als Versuch des Zeichenbenutzers, Erscheinungen zu erfassen, in einer kognitiven Einheit zusammenzufassen und in einen Zusammenhang zu bringen, der eine Abstraktion enthält, die das Wesentliche für den Interpreten, Benutzer oder auch Benutzergruppen (im weiteren repräsentiert durch Akteure) enthält und vom Unwesentlichen im derzeitigen Kontext abstrahiert.

Diese Separation von Gesichtspunkten entspricht dem Herangehen der Semiotik, in der zwischen verwendeter *Syntax*, der unterlegten *Semantik* und der Art der Verwendung (*Pragmatik*) unterschieden wird. In der Semiotik wird unterschieden zwischen Zustands-, Vorgangs-, Tätigkeits- und Handlungsdarstellungen. Syntaktische Formen werden oft in der klassischen SPO-Notation gegeben: Das Subjekt ist Geschehnisträger, Täter, Handelnder, Akteur; das Prädikat ist der Aussagekern; Objekte sind Sinnergänzungen. Außerdem werden freie adverbiale Bestimmungen zur Charakterisierung des Kontextes verwendet. Die Semiotik unterscheidet vier Aspekte: *Syntaktischer Aspekt* zur

[11] Obwohl auch diese Arbeit eine weitgehende Verwendung deutschsprachiger Begriffe bevorzugt, müssen wir beim Begriff "Content" bleiben. Die richtige deutsche Übersetzung führt zum Begriff "Inhalt". Da dieser Begriff in der

Darstellung der Beziehung der Zeichen zueinander; *sigmatischer Aspekt* zur Widerspiegelung der objektiv-realen Wirklichkeit; *semantischer Aspekt* zur Interpretation der Welt durch die Sprache; *pragmatischer Aspekt* zur Konventionalisieurng der sigmatischen, semantischen und syntaktischen Relationen. Der sigmatische Aspekt spielt in der Modellierung keine Rolle mehr, nachdem die Urteile zur Modellierung gefällt wurden.

Ebenso wie in der Modellierung spielen pragmatische Annahmen eine Rolle. So werden z.B. die aktuelle Kommunikationssituation mit der vierstelligen Beziehung zwischen Sender, insbesondere seinem Verständnis, dem Inhalt, der Beziehung zwischen Sender und Empfänger einer Nachricht und dem Empfänger, insbesondere seinem Verständnis mit betrachtet. Ein Analogon der Kommunikationssituation ist die *Anwendungssituation*.

Daraus können wir eine semiotische Triade zu einem Informationssystem ableiten: Der Content bestimmt den syntaktischen Aspekt. Der semantische Aspekt wird durch die Konzeptwelt dargestellt. Der pragmatische Aspekt wird ggf. durch eine Anwendungssituation determiniert und durch eine Begriffslandkarte repräsentiert. In Bild 6 stellen wir die Verbindung zwischen den einzelnen Aspekten

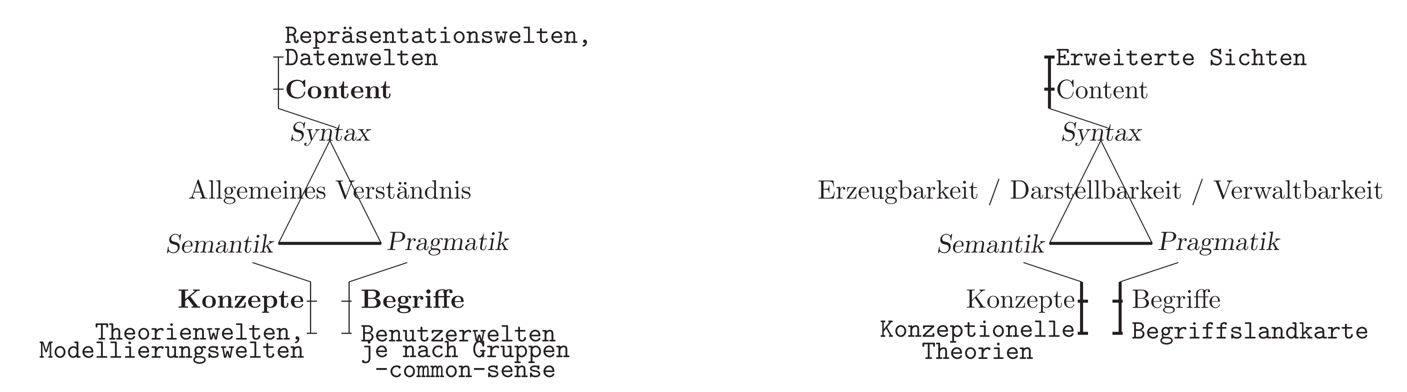


Bild 6: Semiotik-Darstellung von Content, Konzepten und Begriffen

kurz dar.

Damit sind auch die theoretischen Grundlagen von CMS gegeben wie in der folgenden Tabelle:

	Content	Konzepte	Begriffe
Theorie	erweiterte Sichten und Funktionalität	“kleine” logische Theorien	erweiterte Begriffsverbände
Spezifikationsresultat	erweitertes ER-Schema	Konzeptfelder	Begriffslandkarte

Die Assoziation zwischen Content, Konzepten und Begriffen kann erfolgen durch spezielle Abbildungen. Im Rahmen unserer Entwicklung hat es sich als ausreichend erwiesen, dabei wenige ausdrucksstarke Verbindungen der unterschiedlichen Aspekte der Assoziation zu verwenden.

von / zu	Content	Konzepte	Begriffe
Content	Integration	Aufbereitung	Annotation, allgemeine Assoziation
Konzepte	Spezialisierung	Komposition	Verbalisierung, allgemeine Assoziation
Begriffe	allgemeine Assoziation	Untermauerung	Komposition

Content kann z.B. durch eine Datenbankspezifikation wie der folgenden gegeben sein.

```
create table Benutzer (BenutzerID smallint not null,
               FirmaID numeric (18,0) not null
               Vorname varchar (20) null,
               Name varchar (20) null,
               Tel varchar (20) null,
               Zugriff tinyint null, ...)
go
create table BProfile (BPID numeric (18,0) identity (1,1) not null,
               BPName char (100) null,
               BenutzerID smallint null,
```

```
go
create view SysusersBenutzer
as select S1.Name as Login, S2.Name as Gruppe, BP.Name as Profil, BP.Rechte,
               B. Name, B.Vorname, B.Tel, B.Funk, B.FirmaID, S1.GID, S1.UID, ...
          from Sysusers S1 inner join Sysusers S2 on S1.UID <> S2.GID and
               S1.GID = S2.UID left outer join
               Benutzer B on S1.UID = B.BenutzerID left outer join
               BProfile BP on B.BenutzerID = BP.BenutzerID
          where (S1.UID between (select typ_integer from tc_parameter
                    where Name = 'UserAnzeigenAb') and 16380)
go
```

Im allgemeinen wird dies nicht ausreichen. Wir verwenden deshalb erweiterte Sichten, die in den nächsten Kapiteln ausführlich behandelt werden. Sichten müssen um Funktionen erweitert werden, mit denen die Sichten verändert, anders präsentiert und für das Portfolio des Benutzers aufbereitet werden können. Dazu benutzen wir den Definitionsrahmen:

```
generate Mapping : Vars → [temp] output structure
      from   database types   where   selection condition
      represent using   general presentation style
            &   Abstraction (Granularity, measure, precision)
            &   Orders within the presentation
            &   Hierarchical representations
            &   Points of view
            &   Separation
      browsing definition   condition
            &   Navigation
      functions   Search functions
            &   Export functions
            &   Input functions
            &   Session functions
            &   Marking functions
  maintenance functions   maintenance frame
            &   Control (Obligations, Permissions, Restrictions)
```

Konzepte können durch Konzeptnetze dargestellt werden. Konzeptnetze widerspiegeln die drei semiotischen Aspekte Syntax, Semantik und Pragmatik, wobei die Syntax und die Pragmatik durch Kontexte verbunden werden. Konzepte besitzen allgemeine Parameter, die mit einer Wertebereich-Spezialisierungsbeziehung mit Content unterlegt werden können. Diese Parameter können optional oder auch allgemein oder obligatorisch sein. Wir können die Spezifikation von Konzepten mit einem Definitionsrahmen unterstützen oder durch ein Konzeptnetz der Form von Bild 7.

Im allgemeinen wird diese Darstellung durch Konzeptnetze allerdings nicht ausreichend übersichtlich sein. Deshalb kann man nach einer anderen Darstellung suchen. Wir benutzen neben dieser Darstellung auch eine graphische Darstellung durch erweiterte ER-Modelle, bei denen optional Parameter durch eckige Klammern, Identifikationsparameter durch eine Unterstreichung und allgemeine Parameter nicht extra ausgewiesen werden.

Im Falle des Person-Konzeptes können wir drei wichtige Parameter auszeichnen: die Charakterisierung von Personen mit ihren Eigenschaften, die Angabe des Beziehungsumfeldes der Personen und eine Darstellung des Kontextes. Diese Aspekte sind durch entsprechende Logiken unterlegt. Da wir Personen in einer gewissen Allgemeinheit behandeln wollen, wird die Semantik und damit die Theorie mit einer epistemischen, temporalen Logik spezifiziert. Wir betrachten Personen nur im betrieblichen Umfeld und nur aufgrund der Aufgaben, die durch das Informationssystem unterstützt werden. Damit kann man das Person-Konzept holzschnittartig durch eine allgemeine Spezifikation unterlegen der folgenden Form:

$$\text{Person}((_\text{charakteristik},_\text{beziehung},_\text{kontext}), (\Sigma_{\text{Person}}^{\text{DeontTempPL/1}}, \text{M}_{\text{Person}}, \Sigma_{\text{Person}}^{\text{EpistemLogik}}),$$
$$(_\text{betriebsIS},_\text{aufgabenAkteur})).$$

Wir können die Parameter spezialisieren. Eine mögliche Spezialisierung ist die folgende:

$\tau(_\text{beziehung}) = _\text{angestellter} \mathbin{\dot\cup} _\text{partner}$

$\tau(_\text{charakteristik}) = _\text{namen} \mathbin{\dot\cup} _\text{gebDaten} \mathbin{\dot\cup} _\text{identDaten} \mathbin{\dot\cup} _\text{geschlecht}$

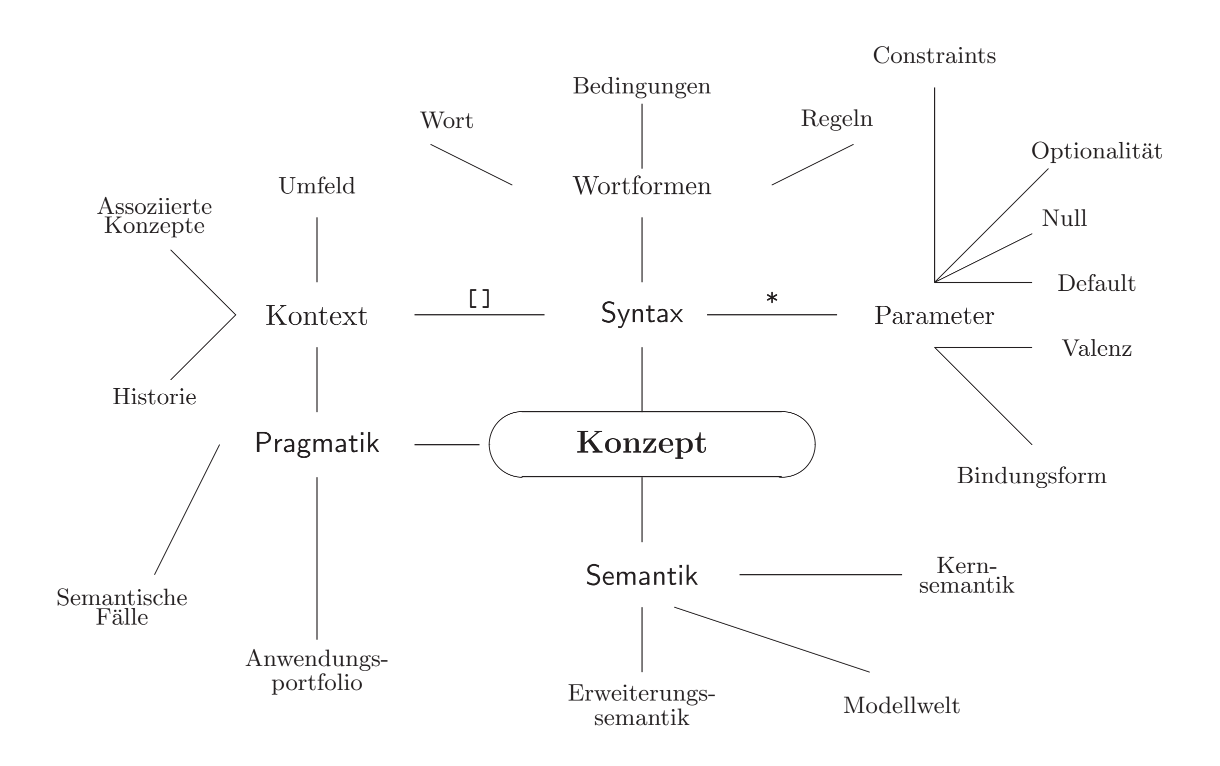


Bild 7: Die Mindmap-Strukturierung der Spezifikation von Konzepten

$\dot{\cup}$ _familie $\dot{\cup}$ _weitereCharakt $\dot{\cup}$ _profil

Die Formeln zur Darstellung der Semantik können unterschiedliche Bereiche der Anwendung abdecken. So können wir z.B. festlegen, daß Personen ihr Geburtsdatum nicht verändern. Eine Person, die *geschieden* ist, war einmal verheiratet. Wir erhalten damit Formeln der folgenden Form, wobei wir uns der deontischen Quantoren F (forbidden), O (obliged) und P (permitted) bedienen:

F(update(Person._gebDaten))

$\alpha_{\text{"geschieden"}}$(_person) $\rightarrow$ $\exists_{past}$ _y
(Beziehung(Ist.Partner._y,Von.Partner._person,Ab,Bis)
$\wedge$ Bis $<$ _today

In analoger Form können wir Adressen spezifizieren:

Adresse((_geographAdr,_kontaktAdr,_historie), ($\Sigma^{PL/1}_{\text{Adresse}}$, M_{Adresse}, $\Sigma^{\text{Qualität}}_{\text{Adresse}}$),
(_betriebsIS,_aufgabenAkteur)).

Die Darstellung der Konzepte kann auch in der Form von ER-Modellen erfolgen. Ein typisches Beispiel wird in Bild 8 vorgestellt.

Konzepte sollen durch Content unterlegt werden können, wobei der Content und seine Struktur variabel sein können, solange sie miteinander verbunden werden können. Wir schränken diese Verbindung durch die Forderung einer Spezialisierungsbeziehung ein:
Die Spezifikation des Content stellt eine Verfeinerung der Spezifikation der zugehörigen Konzepte dar.

Konzepte können miteinander kombiniert werden. So kann z.B. wie in Bild 9 das Konzept *Person* mit dem Konzept *Rolle* und dem Konzept *Adresse* verbunden werden, wobei z.B. nur der *Angestellte* eine interne *Kontaktadresse* und eine externe *Partneradresse* besitzt. Diese Verbindung wird allgemein durch Filter oder "Theta"-Operatoren sichergestellt. Wir können dies durch die Algebra unterstützen und erhalten:

Adressen $\bowtie_{\Theta(\alpha)}$ Personen $\bowtie_{\Theta(\beta)}$ Rollen

Eine Algebra zur Verbindung wird aus der HERM-Algebra abgeleitet. Wir verwenden dabei die HERM-Operationen:

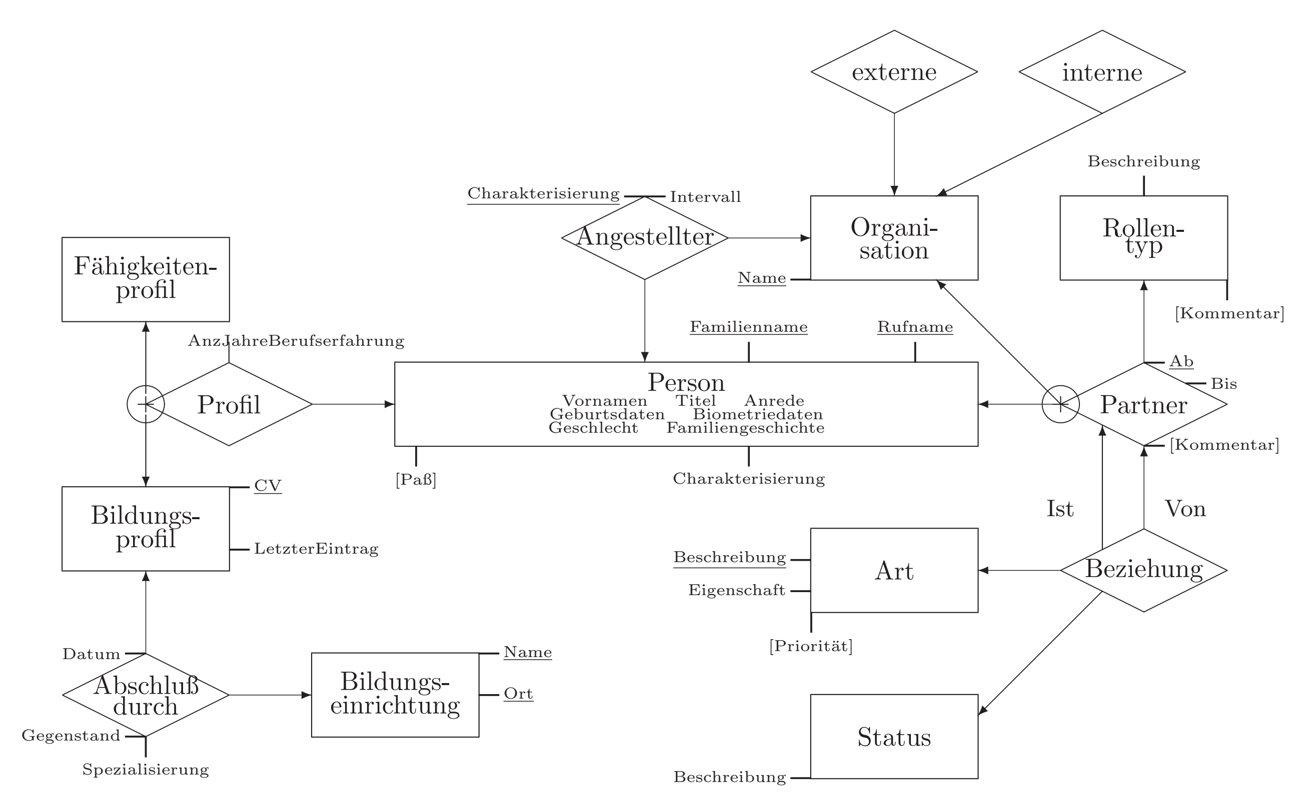


Bild 8: Das Person-Konzept mit obligatorischen, allgemeinen und optionalen Bestandteilen

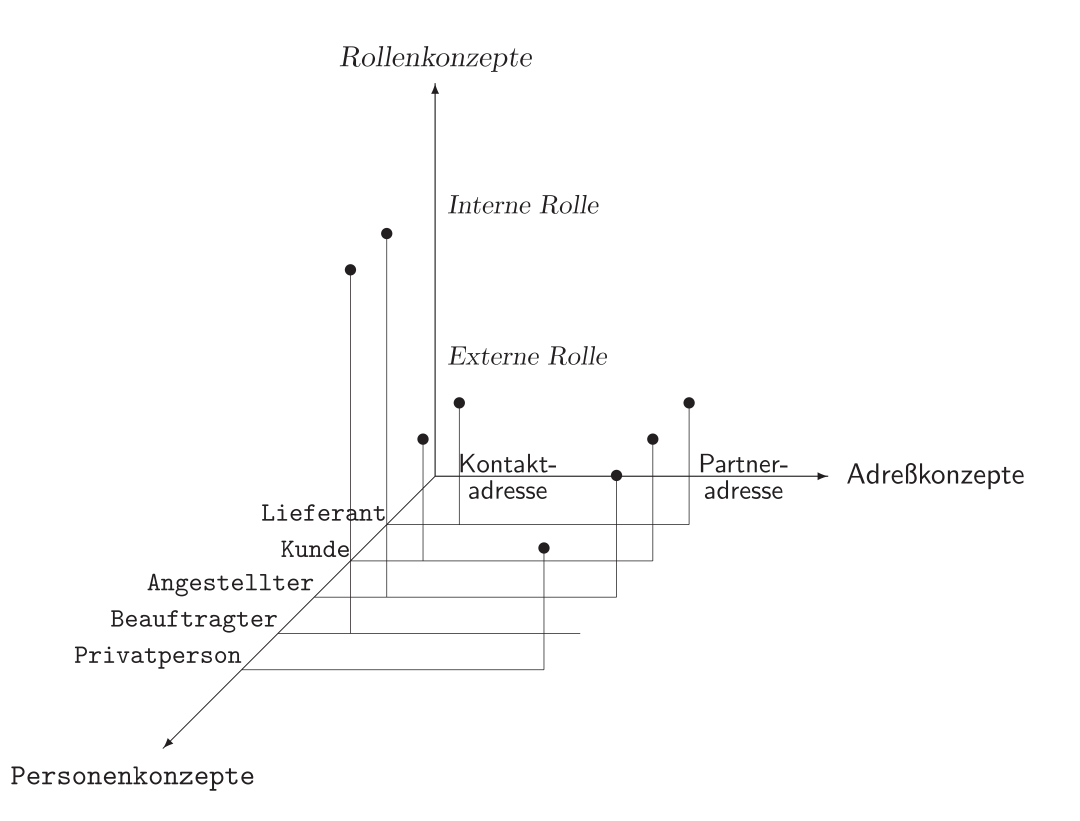


Bild 9: Die Kombination von Konzepten *Person*, *Rolle* und *Adresse*

Eine Spezialisierungsbeziehung zwischen dem Person-Konzept und dem Content erfolgt dann durch Instantiierung der Parameter, Dadurch "paßt" die Sicht zu Person-Content auch zum Person-Konzept. Ein Beispiel ist die Spezialisierung des Parameters _familie oder des Parameters _name:

$$\text{_familie} \rightleftharpoons \begin{array}{l} \mathcal{T}\ (\text{Geburtname, Vater, Mutter}) \\ \text{oder} \\ \mathcal{T}\ (\text{Geburtname, } \{ \text{ Kind } \} \text{)} \end{array}$$

$$\text{_name} \rightleftharpoons \begin{array}{l} \mathcal{T}\ (\text{Vornamen}<(\text{Vorname,use})>\text{, FamName, [GebName,]} \\ \qquad\qquad \text{Titel:}\{\text{AkadTitel}\} \dot\cup \text{ FamTitel)} \\ \text{oder} \\ \mathcal{T}\ (\text{Vorname, Familienname, Spitzname}) \end{array}$$

Begriffe sind i.a. nicht so stark durch Merkmale oder Eigenschaften unterlegt, besitzen häufig eine hohe Ambiguität und sind oft in einer ellipsenartigen Form gegeben. Außerdem werden sie oft metaphorisch verwendet. Begriffe können als Funktionen verstanden werden, die Dinge (im weitesten Sinne) mehr oder weniger abbilden. So wird meist ein Begriff mit einer Menge von Beispielen verbunden, die explizit oder auch abstrakt definiert sein können. Begriffe sind sprachabhängig, meist jedoch nicht reduzierbar auf die Gebrauchsregeln, von denen sie erzeugt werden. Das begriffliche Klassifikationssystem, das eine Sprache unterlegt, ist in hohem Maße Ergebnis eines adaptiven und anwendungskontextgeprägten Sprachwandels.

Begriffe können determiniert werden in der Art und Weise, wie ihre Extension determiniert wird. Sie können scharf begrenzt sein im Sinne "Fregescher Begriffe". Wir bevorzugen diese Form. In der Alltagspraxis werden Begriffe nicht so scharf eingegrenzt. Es ist jedoch Aufgabe der Modellierung, Begriffe so exakt wie möglich Extensionen (Content) und Konzepten zuzuordnen. Begriffe können auch prototypische Begriffe sein oder Familienähnlichkeitsbegriffe.

Ein Beispiel ist das Adreß-Konzept. Wir können mit diesem Konzept unterschiedliche Begriffe verbinden:

- Hauptwohnsitz, Nebenwohnsitz,
- Zustelladresse,
- Anschrift oder Email.

Nicht verbindbar sind dagegen der Begriff Glückwunschschreiben, der Begriff Speicheradresse oder auch der Begriff Eingabe (schriftliche Kundgebung).

Analog stellen wir für das Person-Konzept fest, daß Begriffe wie Mensch assoziierbar sind, nicht aber Figur (Akteur) oder abstrakte Person ("ich für meine Person").

Wir werden im weiteren uns nicht mehr mit Konzepten oder Begriffen auseinandersetzen, da dies den Rahmen dieser Arbeit sprengen würde. Für die Spezifikation von Informationssystemen spielen Begriffe und Konzepte eine untergeordnete Rolle. Wenn wir allerdings eine allgemeinere Architektur, wie z.B. in Bild 10 anstreben, dann kann eine essentielle Verbesserung der Kultur erfolgen. Normalerweise befindet sich ein Benutzer eines Informationssystemes in der SQL-Falle. Er muß sowohl das Schema kennen und verstehen als auch mit SQL seine Anfragen formulieren können. Einfacher und zugleich sinnvoller ist es, dem Benutzer durch eine Assoziation seiner Begriffe mit Konzepten und durch eine Verbindung dieser Konzepte mit Anfrage- und Antwortformen zu unterstützen. Die Anfrageformen können mit dem Datenbankschema ebenso assoziiert werden wie die Antwortformen. Damit erhält ein Benutzer für seine Frage die richtige Antwort aus dem System heraus.

Mit dieser Lösung kann ein Content-Management-System dem Benutzer maximal entgegenkommen.

Die Modellierung von Strukturierung, Funktionalität und Verteilung wird nicht vollständig durch die vorhandene DBMS-Welt unterstützt. Ein Hindernis ist das Sichtenupdate-Problem. Da mit der Erzeugung von Sichten ggf. auch nicht jedes Sichtenobjekt einem Datenbankobjekt zugeordnet werden kann, muß deshalb für eine Modifikation der Datenbank eine andere Funktion zur Verfügung gestellt

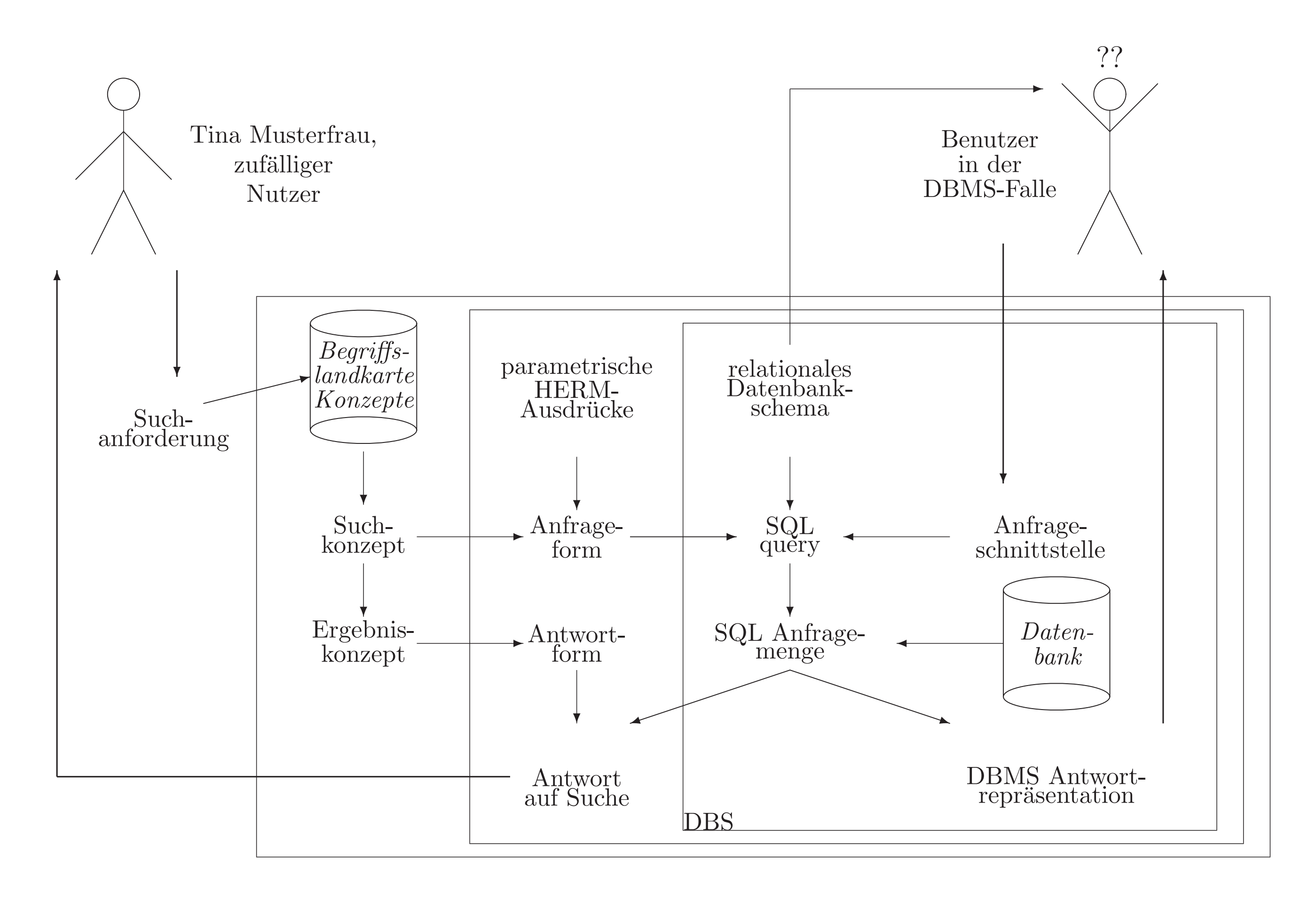


Bild 10: Konzept- und begriffsbasierte Anfrageschnittstellen von Informationssystemen

werden. Deshalb ist die Architektur in Bild 11 eine sinnvolle Alternative, die unserem Vorhaben des integrierten Entwurfes entgegenkommt. Wir unterscheiden damit zwischen Retrieval-Sichten und Modifikationssichten. Dieses Bild zeigt zugleich auch die Unterschiede in der Betrachtungsweise relativ gut auf. Für den Benutzer oder eine Benutzergruppe ist die Anwendung stets lokal. Er nutzt Dialoge, um mit dem Informationssystem bestimmte Aufgaben zu lösen. Dabei werden ihm entsprechende Daten zusammengestellt und übermittelt. Diese Zusammenstellung fassen wir mit einem Container zusammen. Außerdem besitzt dieser Container auch die entsprechende Funktionalität, um den Umgang mit den Daten entsprechend den Dialoganforderungen zu erleichtern. Die Modifikationssichten und die Retrievalsichten sind hierbei entsprechend zusammengefaßt. Das DBMS unterstützt die Anwendung durch die Bereitstellung von Prozessen, die Speicherung der Daten und die Erzeugung und Verarbeitung der Sichten.

Unsere Vorstellungen zur Infrastruktur wird durch Datenbanksysteme gut unterstützt.

Ein gut entwickeltes Datenbanksystem erlaubt die Pflege der Strukturierung und stellt die entsprechende Funktionalität für die Prozesse zur Verfügung. Ein Benutzer sieht ein Informationssystem aus einer anderen Sicht. Ihm wird ein Interaktionsraum zur Verfügung gestellt. Die Benutzung des Systemes findet im Rahmen des Story-Raumes statt. Durch Content-Typen werden der Interaktionsraum und das Datenbanksystemes zu einem Informationssystem verbunden. Wir werden im weiteren sehen, daß Content-Typen eine komfortable Erweiterung des Sichtenkonzeptes für Informationssysteme darstellen.

Ein Aspekt, der nicht vernachlässigt werden kann, bislang aber nur auf strukturellem Niveau behandelt wurde, ist die Verteilung. Es sind dazu eine Reihe von Ansätzen bekannt:

Dienste in verteilten Systemen überwinden die räumliche Trennung von Systemen durch eine Funktionalität zur Datenübertragung und eine zeitliche (gemeinsame) Taktung der Datenspeicherung. Es können in diesem Zusammenhang *Dienstgeber* und *Dienstnehmer* unterschieden werden.

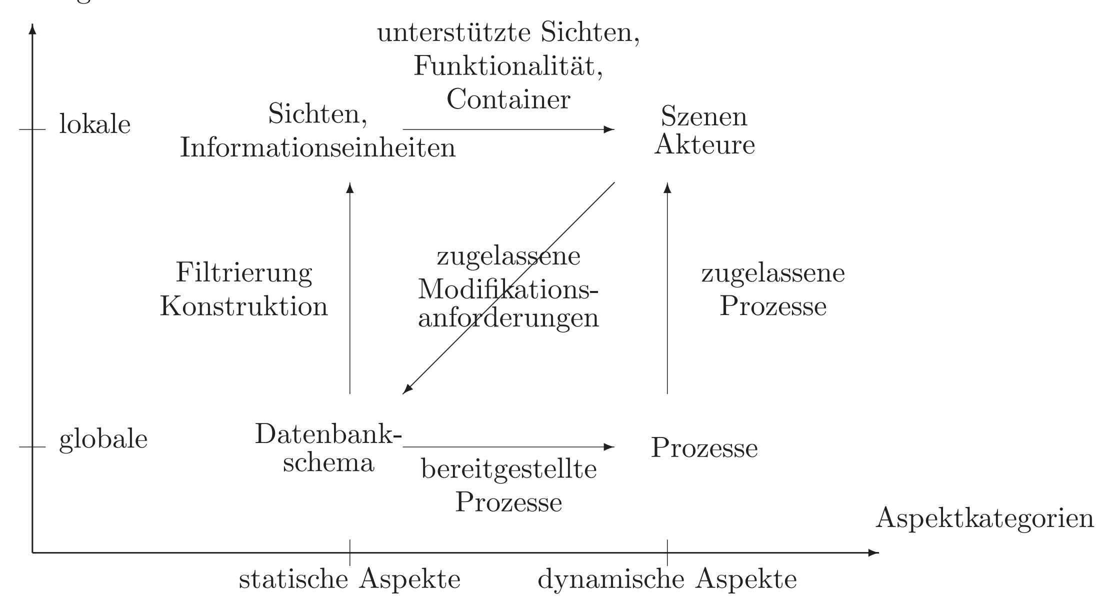


Bild 11: Die Infrastruktur für die integrierte Entwicklung von Informationssystemen

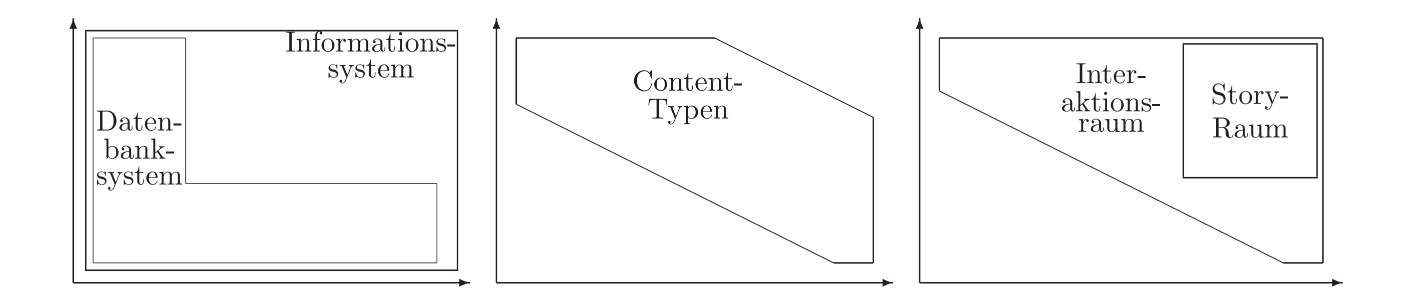


Bild 12: Die Unterstützung von Informationssystemen durch Datenbanksysteme und Content-Typen

unterstützt. Meist basiert diese Architektur auf einer Trennung der Datenverwaltung und des Datenaustausches bzw. der Datenübertragung. Dienste werden charakterisiert durch

eine **Dienstleistungsvereinbarung** als verbindliche Regelung des Dienstverhältnisses,

eine **Sammlung von Funktionen,** die zur Erfüllung des Dienstes abgerufen werden können, und

Dienstmerkmalen zur Darstellung der Qualitätsparameter.

Die Funktionalität des Dienstes und die Dienstmerkmale werden oft als Diensteigenschaften zusammengefaßt.

Verteilte Datenbanksysteme setzen auf Datenbanksystemen auf, erlauben eine Verteilung der Daten durch Partitionierung der Daten, Allokation der Partitionen zu Knoten und eine verteilte Bearbeitung von Daten auf der Grundlage von erweiterten Protokollen für den Abschluß von Transaktionen.

Eine klassische Darstellung der Verteilung wird oft anhand von drei Modellen dargestellt:

Das Kollaborationsmodell oder *Interaktionsmodell* dient der Spezifikation der kommunizierenden Prozesse, ihrer Kommunikation und ihrer Koordination. Es wird meist ein Zeitmodell unterlegt, das auch erlaubt, die Verzögerung der Kollaboration darzustellen.

Das Fehlermodell definiert und klassifiziert die auftretenden Fehler, die Behebungsmöglichkeiten und die Ausführung der Kompensation.

Das Sicherheitsmodell klassifiziert und definiert die Form, mit der Angriffen und Systemgefährdungen begegnet werden kann.

Die Kollaboration führt oft zu einer Einschränkung der Leistung von Systemen. Außerdem kann relativ selten ein globales Zeitkonzept realisiert werden. Deshalb unterscheiden wir auch verteilte Systeme in *asynchrone* und *synchrone* Systeme. Die Kollaboration kann oft durch *Interaktionsdiagramme*, die die Abfolge der Kollaborationsereignisse darstellen, unterstützt werden. Typische Modelle zur Darstellung sind dann Weg-Zeit-Diagramme.

Im Datenbank- und Informationssystementwurf ist jedoch eine größere Vielfalt von Anwendungen darzustellen. So sind z.B. e-Business-Anwendungen mit den Methoden der OSI-Schichtung, mit einfachen Diensten und auf der Grundlage von einfachen Austauschprotokollen nur partiell darstellbar. Deshalb wird versucht, über mehrdimensionale Strukturierung, wie z.B. in [ALSS03], mit den Dimensionen Datenübertragungssystem und Datenverwaltungssystem und den klassischen Schichten des OSI-Modelles (Bitübertragung, Sicherung, Netz und Vermittlung, Transport, Anwendung wie z.B. Middleware) und des verallgemeinerten 5-Schichten-ANSI-Modelles (extern, intern, physisch, Segment, Datei) mit Erweiterungen zu einer dritten Dimension, die durch HW-SW-Systeme determiniert wird, sich eine Übersicht zu verschaffen. Anwendungssysteme wie Middleware-Systeme unterstützen die Kopplung von Informationssystemen. Middleware kann bei der Überwindung der Heterogenietät durch entsprechende Transformationsmechanismen zur Typkonversion und Aufrufumformung unterstützen. Diese Umformungen können auf der Grundlage von Regeln vorgenommen werden. Stummel-Objekte werden dienstnehmerseitig erzeugt und Gerüst-Objekte werden dienstgeberseitig bereitgestellt. Es wird ein Funktionsaufruf wie in Bild 13 realisiert.

Die Vermittlung von Dienstgebern basiert auf einem Vermittlungsdienst, einem Namendienst mit einem Namensraum und einer entsprechenden Navigationsunterstützung. Innerhalb des verteilten Systemes werden Dienste aktiviert und beendet, Lasten verteilt, die Sicherheit und die Persistenz garantiert und eine verteilte Transaktionsverwaltung mit einem Ressourcen-Verwalter, einem Synchronisationsdienst und einem Transaktionsverwalter unterstützt. Typische Architekturen für Middleware-Systeme sind CORBA und Web-Dienste.

Wir sind jedoch mehr an einer konzeptionellen Modellierung der Verteilung interessiert. Eine CORBA- oder Web-Dienste-Spezifikation ist eine physische oder logische Umsetzung. Deshalb verwenden wir das Diensteverwaltungssystem mit den entsprechenden Schnittstellen und Protokollen auf dem Implementationsniveau. Zur Spezifikation der Verteilung abstrahieren und verallgemeinern die Ansätze zur Verteilung:

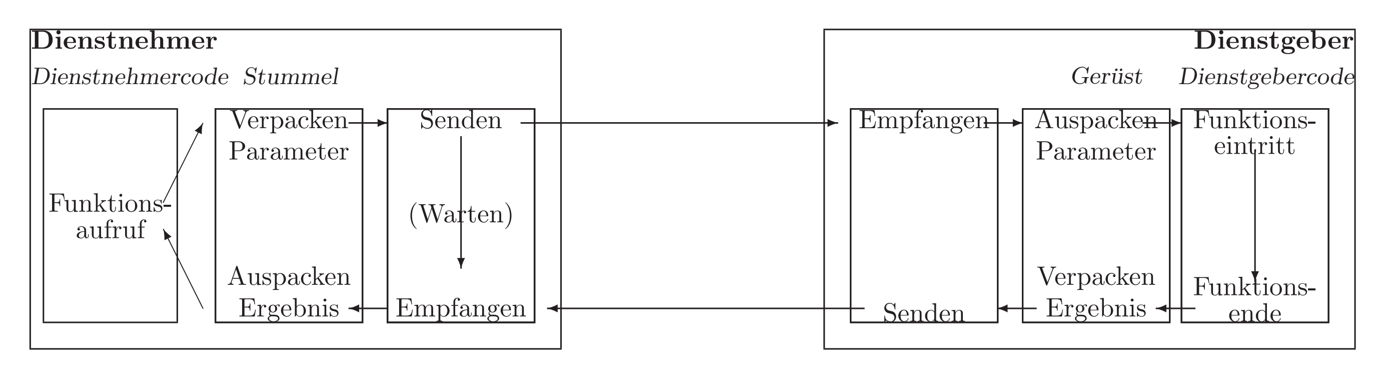


Bild 13: Entfernter Funktionsaufruf mit einer Schichtung [ALSS03]

Die **Verteilung** ist gegeben durch eine Spezifikation der Dienste des Kollaborationsrahmens und der Architektur. Dienste setzen auf Sichten-Suiten auf. Der Kollaborationsrahmen faßt die Kommunikation, die Koordination und die Kooperation zusammen. Die Architektur stellt den Zusammenhang der Komponenten dar.

Dienste $\mathcal{S} = (\mathcal{I}, \mathcal{F}, \Sigma_{\mathcal{S}})$ sind gegeben durch die Informationseinheiten, durch das Dienstverhalten, und durch den Dienstvertrag, insbesondere die Qualitätsparameter.

Informationseinheiten $\mathcal{I} = (\mathcal{V}, \mathcal{M}, \Sigma_{\mathcal{T}})$ bestehen aus Content-Typen der Sichten-Suite, einem Informationseinheit-Manager und einer Menge von Regeln zur Darstellung der Kompetenz.

Das Dienstverhalten wird

- innerhalb der Aktionsschicht durch *Vereinbarungen zur Dienstgüte*,
- innerhalb der konzeptionellen Schicht durch *konzeptionelle Eigenschaften* und
- innerhalb der Implementationsschicht durch *Dienstgüteeigenschaften der Implementation*

angegeben.

Der Dienstvertrag legt die Rahmenbedingungen des Dienstes fest.

Das Vertragsschema stellt die Bedingungen des Vertrages dar. Insbesondere werden Parameter wie

- das *Benutzungsmodell* (mit den *Akteuren*, ihren *Beziehungen*, *Rollen* und *Rechten*),
- das *Zeitmodell*,
- der *Vertragskontext* und
- die *vertraglich vereinbarte Qualität*

spezifiziert.

Qualitätsparameter der Dienste sind je nach Abstraktionsniveau

- innerhalb der Aktionsschicht Eigenschaften wie *Allgegenwart* (ubiquity) und *Sicherheit*,
- innerhalb der konzeptionellen Schicht Eigenschaften wie *Bedeutungstreue* und *Konsistenz* und
- innerhalb der Implementationsschicht Eigenschaften wie *Dauerhaftigkeit*, *Performanz*, *Robustheit* und *Skalierbarkeit*.

Der Kollaborationsrahmen ist durch die Darstellung der verschiedenen Facetten der Kollaboration spezifiziert:

Der Kommunikationsrahmen legt die Art der Kommunikation und die benutzten Austauschmechanismen fest.

Der Kooperationsrahmen bestimmt die Art des Zusammenwirkens der unterschiedlichen Akteure und Komponenten im Rahmen des Portfolios bzw. der Arbeitsprozesse.

Der Koordinationsrahmen bestimmt die Synchronisation der Kollaboration, die Organisation und die Aufgabenverteilung.

Die Facetten der Kollaboration werden durch jeweils drei Teilspezifikationen angegeben:

Der Diskurs bestimmt den Ablauf der Kollaboration. Er basiert auf den anderen drei Bestandteilen des Co-Design-Ansatzes:

Die Daten werden zu Content verdichtet und durch Sichten über dem Datenbanksystem angegeben.

Die Funktionalität wird durch Angabe der unterstützenden Systemfunktionen dargestellt.

Die Interaktivität basiert auf dem Storyboard der Anwendung.

Der Stil der Kollaboration legt die vertraglichen Vereinbarungen fest. Er wird durch

- die *Unterstützungsprogramme* wie Sitzungsverwaltung, Benutzerverwaltung und der Abrechnung,
- den *Datenzugriffsrahmen* mit den Varianten zwischen broadcast und peer-to-peer, dem gemeinsamen Benutzen von Ressourcen und den Zugriffsformen und
- die *Art* wie peer-to-peer- oder der Ereignis- oder der Komponentenkollaboration sowie
- den *Koordinations-Workflow* mit den Partnerbeziehungen, dem Diskurstyp, dem Namensraum und den Workflow-Regeln

determiniert.

Die Kollaborationsarchitektur bzw. das Kollaborationsmuster verbinden die Komponenten. Das Kollaborationsmuster ist eine Verallgemeinerung der Protokolle mit einer Darstellung der Partner, ihrer Aufgaben, ihrer Rollen und Rechte. Wir unterschieden zwischen

- *Proxy-Kollaboration*,
- *Broker- bzw. Trader-Customer-Kollaboration*,
- *Client-Dispatcher-Kollaboration*,
- *Publisher-Subscriber-Kollaboration* und
- *Model-View-Controller-Kollaboration*.

Die Architektur kann als Verallgemeinerung der Architektur verteilter Datenbanksysteme angesehen werden. Architekturen *föderierter Systeme*, von *Datenbank-Farmen* und *inkrementellen Datenbanksystem-Suiten* basieren auf einer Trennung in lokale und globale Komponenten und auf der expliziten Spezifikation der Austauschbeziehungen zumindest für die Strukturierung von Objekt-Suiten, mitunter auch die Funktionalität von Objekt-Suiten.
Architekturen können durch entsprechende Austauscharbeitsplätze unterstützt werden.

Unsere konzeptionelle Darstellung kann auf die logische und physische Ebene abgebildet werden durch:

Abbildungsregeln zur Transformation von Sichten,

Abbildungsregeln zur Erzeugung der Architektur,

Abbildungsregeln zur Erzeugung des Kollaborationsmodelles,

Abbildungsregeln zur Erzeugung des Fehlermodelles und

Abbildungsregeln zur Erzeugung des Sicherheitsmodelles.

3 Das Abstraktionsschichtenmodell

Gebraucht der Zeit, sie geht so schnell von hinnen!
Doch Ordnung lehrt Euch Zeit gewinnen.
Mein teurer Freund, ich rat euch drum
Zuerst Collegium Logicum.
J.W. Goethe, Faust, Erster Teil, Studierzimmer, Mephistopheles

Wir erhalten aus Anforderungen der vorigen Kapitels die Aufgabe, Strukturierung, Funktionalität, Dialoge, Verteilung und Sichten auf eine Datenbank im Zusammenhang zu entwerfen. Vereinfachend ist dabei, daß die Dialoge auf den Sichten und den Prozessen aufsetzen und daß die Sichten in das Schema einbindbar sein sollen. Die Prozesse werden damit über diese Einbindung in das Schema auch für die Sichten benutzbar. Damit erhalten wir ein *Entwurfsviereck*, bestehend aus der Datenspezifikation, der Funktionsspezifikation, der Verteilungsspezifikation und der Dialogspezifikation.

Das Zachman-Modell zur Separation von Aspekten

Durch Zachman wurden Ende der achtziger Jahre allgemeine Modellierungsregeln eingeführt, die mit dem Abstraktionsschichtenmodell verallgemeinert werden:

- Es werden verschiedene Dimensionen der Entwicklung unterschieden:
  - Die *Dimension der statischen Aspekte* stellt Strukturierung der Daten und die Sichten dar.
  - Die *Dimension der dynamischen Aspekte* soll die Funktionalität und die Interaktivität der Anwendung repräsentieren.
  - In der *Verteilungsdimension* wird die Lokalität der Strukturen und Prozesse dargestellt.
  - Die *Benutzerdimension* dient der Darstellung des Systemes aus Benutzersicht einschließlich der Organisationsmodelle.
  - In der *Zeitdimension* wird die Entwicklung der Anwendung dargestellt.
  - Mit der *Motivationsdimension* erfolgt eine explizite Darstellung der Umstände, Ziele und Motive für die einzelnen Aspekte der Anwendung.
- Jede der Dimensionen verfügt über ein einfaches und eindeutiges Basismodell.
- Jede der Dimensionen repräsentiert genau eine Sichtweise auf die Anwendung.
- Jedes abstrakte Objekt wird nur einmal repräsentiert.
- Entwurfsprodukte sind aufgrund ihrer Architektur und Entwurfsgeschichte rekursiv oder iterativ aufgebaut.

alität	statische Aspekte	dynamische Aspekte				
Planer	Klassen der Geschäftsdaten	Klassen der Geschäftsprozesse	Hauptgeschäftsorte	Hauptorganisationseinheiten	Hauptgeschäftsereignisse	Hauptzi
Besitzer	Geschäftsobjekte	Geschäftsprozesse	Geschäftsorte	Organisationseinheiten	Geschäftsereignisse	Geschäf
Entwerfer	ER-Typen für Datenbankstruktur und Sichten	Anwendungsfunktionen, Stories	Knotenfunktionen	Rolle	Systemereignisse	Kriteriu
Programmierer, Anwendungsentwickler	Logisches Schemata und Sichten	Prozesse, Szenen, Dialogschritte	System-Software	Akteure	Ausführungszeit	Bedingu
Komponentenlieferant	Datenkomponente	Prozeßkomponente	Adressen	Akteure	Interrupt- und Berechnungsmodelle	Teilbedingung
Resultate	Strukturen, Content-Typen, Container, Sichten	Prozesse, Szenario im Story-Raum	Netze, Allokationen	Organisationsspezifikation	Ausführungsplan	Strategi
Modelle	Computermodell			Ausführungsmodell		

Das verallgemeinerte Zachman-Modell mit den Sichtweisen der beteiligten Seiten und dem Zusan schichtenmodell

	Ziel	**Produkt**	**R**
Planer	Beschreibung des Spielraumes	Beschreibung des Spielraumes	F
Besitzer	Reales Produkt	Geschäftsmodell	S
Entwerfer	Beschreibung des abstrakten Produktes	System-Modell	U
Implementierer	Beschreibung des Produktes und seiner Verwendung	Technologie-Modell	S V
Komponentenlieferant	Beschreibung der Komponenten	Out-of-context-Modelle	Ir

Das Entwicklungsmodell in den unterschiedlichen Sichtweisen

Mit dieser Verallgemeinerung wird die Mitwirkung unterschiedlicher Personen zu unterschiedlichen Zeiten im Entwicklungsprozeß sichtbar:

Planer in der Motivationsschicht: Durch den Systemplaner wird eine Analyse des gegebenen Zustandes und eine Zielbestimmung für die gesamte Anwendungsentwicklung vorgenommen.

Besitzer in der Motivationsschicht: Durch den Besitzer werden die Randbedingungen für die Entwicklung vorgegeben.

Entwerfer in der konzeptionellen Schicht: Ein Entwerfer ist hauptverantwortlich in der konzeptionellen Schicht, zugleich allerdings Partner der anderen Personen in allen anderen Schichten.

Programmierer und Anwendungsentwickler in der Implementationsschicht verwenden die Entwurfsdokumente zum Erstellen der Programme. Änderungen der Entwurfsdokumente sind abzustimmen.

Komponentenlieferant in Abhängigkeit vom Entwicklungsmodell: Das Komponentenmodell ist orthogonal zu den anderen Entwicklungsmodellen und wird deshalb auch in die anderen Entwurfsdokumente integriert. Je nach Abstraktionsschicht erfolgt eine unterschiedliche Einbindung.

Das Zachman-Modell der Rollen während der Entwicklung von Informationssystemen ist noch relativ grob. Wir können feiner unterscheiden z.B.

Rollen aus dem Umfeld (Genehmigungsbehörden, Einspruchsberechtigte, Öffentlichkeit),

Rollen der Bestellung ('Bauherr', Eigentümer, Finanzgeber (Investor, Finanzierender, Subventionsgeber), Betreiber (Verwaltung, Erhaltung), Benutzer, Projektleiter, Besteller, Berater),

Rollen der Lenkung: (Gesamtleitung, Leitung Projektierung, Leitung Programmierung, Leitung Administration, Leitung Infrastruktur),

Rollen der Gestaltung (Projektierung, Architekt, Berater) und

Rollen der Ausführenden (Entwerfer, Graphikdesigner, Programmierer).

Das Zachman-Modell verdeutlicht unterschiedliche Abstraktionsschichten mit unterschiedlicher Spezifikation und unterschiedlicher Detaillierung. Ein integrierter Entwurf muß deshalb auch unterschiedliche Detaillierungsgrade ermöglichen. Günstig ist, wenn die Entwurfsdokumente aufeinander Bezug nehmen bzw. eine Untersetzung der Entwurfsdokumente der nächsthöheren Schicht wie in Bild 14 darstellen.

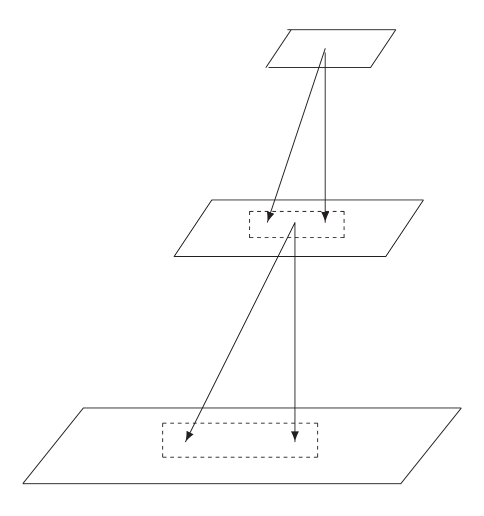


Bild 14: Entwurfseinheiten auf verschiedenen Abstraktionsebenen

Gleichzeitig beobachten wir, daß drei Dimensionen in der Modellierung auseinander gehalten werden müssen:

Abstraktionsschicht: Die Schichtung sollte hierarchisch wie in Bild 14 erfolgen, damit die Entwurfsdokumente zueinander einfach in Beziehung gesetzt werden können.

Architektur der Komponenten: Können die Komponenten der Anwendung separiert werden, dann kann auch eine Architektur der Komponenten mit expliziter Darstellung ihrer Zusammenhänge erfolgen.

Versionen der Entwicklungsresultate: Jedes Entwurfsdokument kann im Verlaufe der Entwicklung revidiert werden. Deshalb sollte man eine explizite Pflege von Versionen in den Entwurfsprozeß integrieren.

Diese drei Dimensionen spannen einen Entwicklungsraum wie in Bild 15 auf.

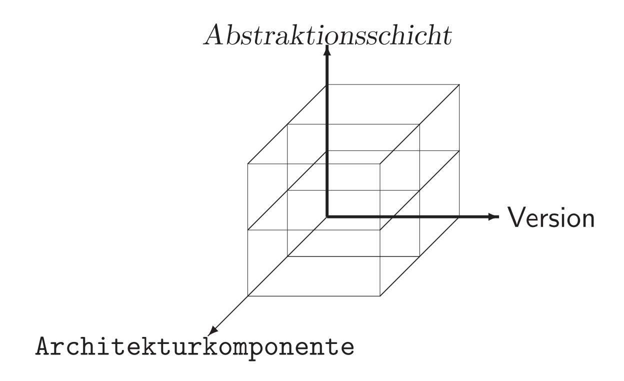


Bild 15: Entwicklungsdimensionen für dei Entwurfsdokumente

Das Abstraktionsschichtenmodell zur integrierten und abgestuften Entwicklung

Wir betrachten explizit unterschiedliche Abstraktionsschichten und integrieren die Darstellung der Architektur der Anwendung und die Versionierung explizit in die einzelnen Entwurfsschritte. Damit unterscheiden wir folgende Schichten:

die Motivationsschicht zur Spezifikation der Ziele, der Aufgaben und der Motivation der Informationssystemanwendung,

die Geschäftsprozeßschicht zur Spezifikation der Geschäftsprozesse, der Ereignisse, zur Grobdarstellung der unterlegten Datenstrukturen und zur Darstellung der Anwendungsstory,

die Aktionsschicht zur Spezifikation der Handlungen, der Detailstruktur der Daten im Sinne eines Vorentwurfs, zur Darstellung eines Sichtenskeletts und zur Darstellung von Szenarien zu den einzelnen Anwendungsstories,

die konzeptionelle Schicht zur Darstellung der Prozesse, des konzeptionellen Schemas, der konzeptionellen Sichten und der Dialoge in zusammenhängender Form,

die Implementationsschicht zur Spezifikation der Programme, der physischen und logischen Schemata, der externen Sichten und zur Darstellung der Inszenierung.

Die Motivationsschicht kann als strategische Schicht aufgefaßt werden. Es werden alle strategischen Entscheidungen zum Informationssystem getroffen. Die Geschäftsprozeßschicht wird oft auch als Anforderungsspezifikationsschicht bezeichnet. Im Rahmen dieser Schicht werden neben den Anforderungen jedoch auch konkrete Entscheidungen zur Realisierung getroffen, so daß wir diese Schicht zur Spezifikation der Anforderungen, der pragmatischen Annahmen, der Systemumgebung und der Systemorganisation und -architektur erweitern müssen. Die Aktionsschicht ist mit dem Abstraktionsschichtenmodell eingeführt worden, um eine explizite Darstellung der Anwendungsszenarie vornehmen zu

können. Im klassischem Systementwurf wird diese Schicht meist übergangen und zu einem späteren Zeitpunkt durch entsprechende Sichten-Suiten hinzu gefügt. Damit entsteht ein Systembruch, den wir mit der expliziten Darstellung vermeiden können.

Die Betrachtung der physischen Realisierung ist keine Aufgabe des Informationssystementwurfes und wird ebenso wie die Pflege- und Einführungsschicht in diesem Buch nicht behandelt. Die Verteilungs- und die Sicherheitsaspekte sind orthogonale Aspekte und werden mit den Entwicklungsschritten verflochten.

Das Abstraktionsschichtenmodell in Bild 16 erlaubt eine Entwicklung von Informationssystemen im Zusammenhang. Wir können ein schichtenorientiertes Vorgehensmodell ebenso wie ein Modell anwenden, das sich zuerst auf einen der Aspekte orientiert.

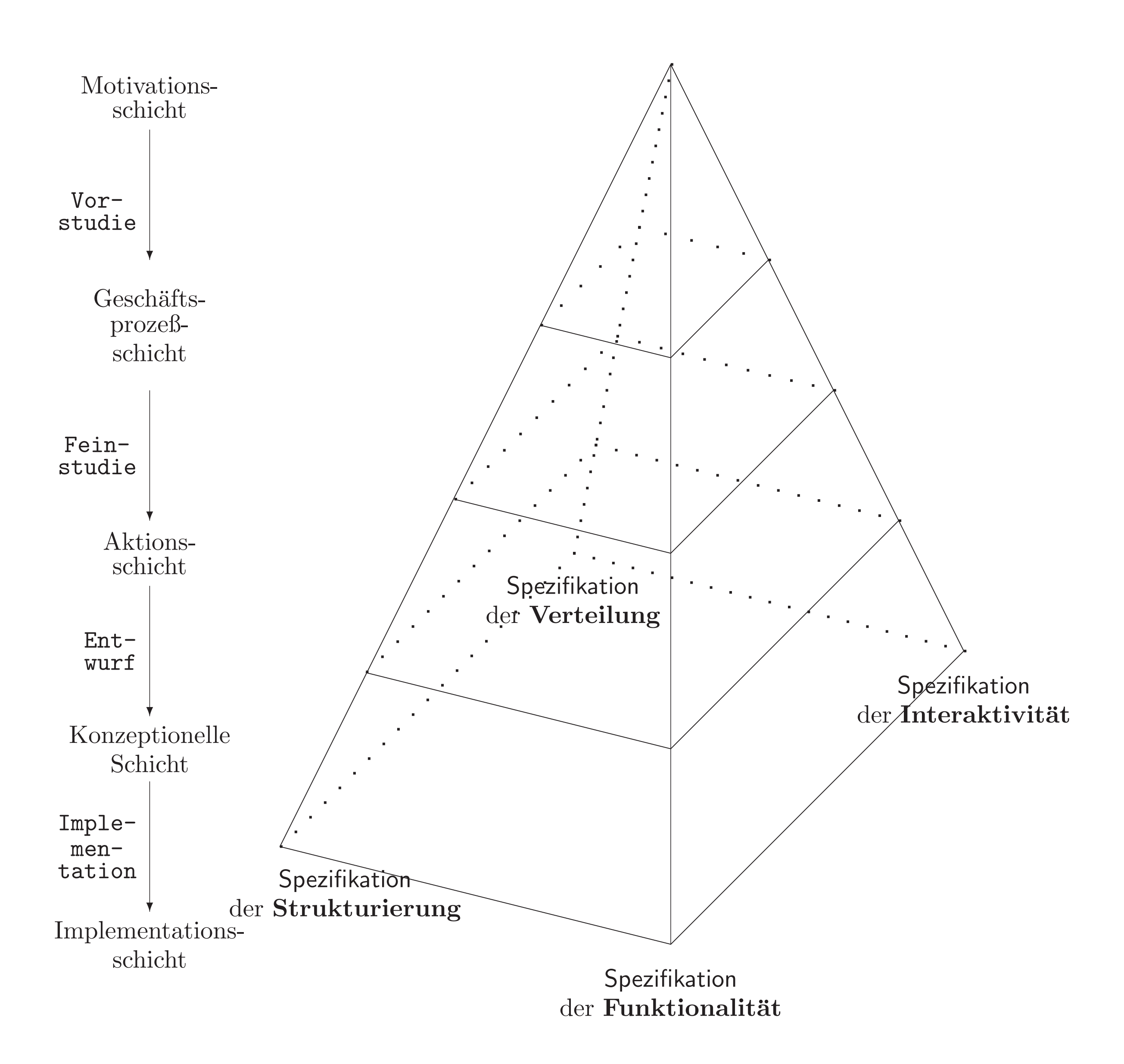


Bild 16: Das Abstraktionsschichtenmodell des Informationssystem-Entwicklungsprozesses

Die Spezifikationssprachen können sich für die Schichten und die einzelnen Spezifikationsteile stark unterscheiden. Eine solche Sprachvielfalt ist jedoch nicht immer angebracht. Wir können aber einen Sprachmix verwenden, der sich mit jeder weiteren Schicht immer stärker auf die formalen Teile orientiert. Vorstellbar und praktikabel ist ein Sprachmix aus natürlichsprachigen Äußerungen, Formulartechniken und formalen Darstellungsmitteln wie Diagrammen zur Darstellung der Datenstrukturen

die Implementationsschicht benötigen wir eine formale Darstellung mit exakt definierter Semantik, für die konzeptionelle Schicht ist dies ebenso notwendig. Wenn wir uns für einen Sprachmix entscheiden, dann sollten wir in jedem Fall die Abbildbarkeit der Konstrukte von Schicht zu Schicht garantieren können.

Auf die natürliche Sprache sollte schon aufgrund des ihr innewohnenden Potentials keinesfalls verzichtet werden. Formulartechniken sind eine Vorstufe der formalen Darstellung. Formale Techniken wie ER-Modelle oder CSP-Modelle sind für den direkten Anwender weniger geeignet, sind aber - mit einer entsprechenden Semantik versehen - sehr gut zur Darstellung in der konzeptionellen Schicht geeignet.

Wir werden im weiteren zuerst einmal die einzelnen Spezifikationsteile im Abstraktionsschichtenmodell untersuchen. Dabei können wir auf die Erkenntnisse, die in den vorangegangenen Kapiteln dargestellt sind, zurückgreifen. Anschließend zeigen wir, wie ein schichtenorientiertes Vorgehensmodell im Sinne eines allgemeinen Top-down-Modelles sinnvoll, einfach und im Zusammenhang angewandt werden kann.

Mit unserem Vorgehen entsteht für eine etwas umfangreichere Anwendung mit etwa 500 Entity-, Relationship- und Cluster-Typen im ersten Schritt ein kurzes (sechsseitiges) Essay mit der Beschreibung der Ideen und Motive, ein längeres (30-seitiges) Treatment (Lastenheft) zur groben Darstellung der Strukturen, Prozesse, Dialoge und Zusammenhänge, ein (100-seitiges) Rohbuch (Pflichtenheft) zur Darstellung der Aktionen, Vorentwürfe, Handlungen, Sichtenskelette, Szenarien, ein (200-seitiges) Buch zur Darstellung des konzeptionellen Entwurfes und ein (500-seitiges) Werk zur Darstellung der Implementation. Diesem Vorgehen kann entgegengehalten werden, daß ein von Intuitionen geprägter Entwicklungsprozeß eher geeignet ist, ein Ziel zu erreichen. Damit kann ein einfacher Entwurf entstehen, der in einem Lehrbuch etc. dargestellt werden kann, nicht aber ein komplexerer Datenbankentwurf. Wie sehr ein intuitiver Stil gepflegt werden kann, hängt auch von der Professionalität der Entwerfer ab, die - wie die $(DB)^2$ bzw. ID^2 community zeigt - z.T. umfangreiche, verarbeitete, bewußte und vor allem unbewußte Kenntnisse über die Strukturen, Prozesse und Dialoge einer Anwendung besitzen.

In den nächsten Teilkapiteln stellen wir zuerst die Datenspezifikation, die Funktionsspezifikation und die Sichtenspezifikation in aller Kürze vor. Anschließend führen wir exemplarisch die Dialogspezifikation detailliert ein. Da für die ersten drei Spezifikationen bereits viele Untersuchungen existieren, für die letzte aber kaum Material existiert, versuchen wir damit auch zugleich eine Lücke in der Datenbankliteratur zu schließen.

Resultate der Entwicklung auf unterschiedlichem Abstraktionsniveau

Das Abstraktionsschichtenmodell erlaubt die Darstellung der Entwicklungsresultate auf unterschiedlichem Abstraktionsniveau. Wir folgen hier im wesentlichen dem induktiven Ansatz zur Beschreibung. Damit ist jedes Resultat aus jeder Sichtweise (Strukturierung, Funktionalität, Interaktivität, Unterstützung der Interaktivität) als generelle Einheit oder Basiseinheit spezifizierbar. Resultate der Entwicklung der Informationssystemanwendung sind:

Produkte zur Darstellung der Strukturierung sollen die Strukturierung der Daten auf unterschiedlichem Abstraktionsniveau beschreiben. Wir nutzen dazu eine Separation der Spezifikation in

Schema zur Beschreibung der gesamten Strukturierung und

Daten-Typ zur Beschreibung der einzelnen Struktur und der Integritätsbedingungen.

Produkte zur Darstellung der Funktionalität sollen eine Darstellung der Funktionsaspekte ermöglichen. Wir unterscheiden

Workflows zur Darstellung der Folgen von

Prozessen der Anwendung

Produkte zur Darstellung der Interaktivität sollen eine Beschreibung der Anwendung aus der Sicht der Benutzer ermöglichen. Deshalb wird die Interaktivität als Raum von Handlungsabläufe der Benutzer oder ihrer Abstraktionen als Akteure, d.h. als Story-Raum beschrieben. Dieser Story-Raum fußt auf

Szenen zur Beschreibung eines generellen Schrittes der Anwendung und auf

Dialogschritten zur Beschreibung der einzelnen Aktionen.

Produkte zur Darstellung der Unterstützung der Verteilung sind im Rahmen von Anwendungen der Informationssysteme

Sichten auf die Datenbanksysteme,

Dienste zur Bereitstellung der erweiterten Sichten und deren

Austauschrahmen.

Wir wollen diese Entwicklungsresultate auf unterschiedlichem Abstraktionsniveau darstellen. Wir können jeweils die Resultate mit der Abstraktionsschicht verbinden. Dann sind die Abstraktionsschichten mit folgenden Entwicklungsresultaten verbunden:

Motivationsschicht mit dem ***Lastenheft,***

Geschäftsprozeßschicht mit dem ***Pflichtenheft,***

Aktionsschicht mit der ***Aktionsspezifikation*** und den vier Aspekten Anwendungsschema, Nutzer-Maschine, Storyboard und Aktionssichten-Suite,

Konzeptionelle Schicht auf Grundlage der ***konzeptionellen Spezifikation*** und der Beschreibung der vier Aspekte durch ER-Schema, Workflow-Maschine, Drehbuch und Sichten-Suite,

Implementationsschicht auf Grundlage der ***logischen Spezifikation*** und einer Beschreibung der vier Aspekte durch logisches Schema, Datenbank-Maschine, Inszenierung und logische Sichten-Suite.

Demzufolge können wir die Entwicklungsprodukte für die entsprechenden Abstraktionsschichten wie auf der folgenden Seite darstellen.

	Motivationsschicht	Geschäftsprozeßschicht	Aktionsschicht	Konzeption
Schema	Schema durch Konzeptlandkarte	Schema durch Skizze	Schema durch Anwendungsschema	konzeption Schema
Datentyp	Datentyp durch Konzept	Datentyp durch groben Typ	Datentyp durch Anwendungstyp	konzeption typ
repräsentiert im	Datenteil des Lastenheftes	Datenteil des Pflichtenheftes	Anwendungsschema	ER-Schema
Workflow	Workflow durch Produktfunktionalität	Workflow durch Geschäftsprozesse	Workflow durch Handlungen	konzeption flow
Prozeß	Prozeß durch Produktfunktion	Prozeß durch Arbeitsschritt	Prozeß durch Aktion	konzeption
repräsentiert im	Funktionenteil des Lastenheftes	Funktionenteil des Pflichtenheftes	Nutzermaschine	Workflow-N
benutzt in			Content-Typen	Content-Ty
Szene	Szene im Anwendungsgebiet	Szene in einer Story	Szene im Plot	Szene im des Drehbu
Dialogschritt	Dialogschritt durch Anwendungsschritt	Dialogschritt durch Anwendungsereignis	Dialogschritt durch Thema der Anwendung	konzeption alogschritt
repräsentiert im	Diskursteil des Lastenheftes	Handlungsrahmenteil des Pflichtenheftes	Storyboard	Drehbuch
benutzt direkt			Content-Typen	Content-Ty
Sicht	Produktdatensicht	Skizze der Sicht	Skelett der Sicht	Schema de
Typ der Sicht	Produktdatentyp	Typ durch ontologische Einheit	Kerntyp	konzeption
repräsentiert im	Sichtenteil des Lastenheftes	Sichtenteil des Pflichtenheftes	Aktionssichten-Suite	Sichten-Sui
benutzt in			Content-Typen	Content-Ty

Entwicklungsprodukte auf den entsprechenden Abstraktionsschichten

Ein Vorgehensmodell

Wir können mit dem Abstraktionsschichtenmodell zur Entwicklung von Informationssystemen eine Reihe verschiedener Entwicklungsmodelle unterstützen:

In der Strukturierungsorientierten Entwicklung wird zuerst die Datenbank-Struktur weitestgehend entwickelt. Darauf aufbauend werden die Prozesse und die Sichten und abschließend die Präsentationskomponente entworfen und implementiert. Diese Vorgehensweise entspricht dem klassischen Entwicklungsansatz, hat aber den Nachteil einer hohen Modifikationsrate aller vorher erstellten Dokumente.

In der prozeßorientierte Entwicklung wird zuerst die Funktionalität der Anwendung entworfen und prototypisch realisiert. Danach werden die entsprechenden Datenstrukturen entwickelt und abschließend die Präsentationskomponente und die entsprechenden Sichten. Dieser Zugang wird im Software-Engineering präferiert, entspricht aber selten den Gegebenheiten der Entwicklung von Informationssystemen.

Interaktionsraum-determinierte Entwicklung: Es werden zuerst die Stories und Szenarien der Anwendung abgenommen. Auf dieser Grundlage werden die entsprechenden Medientypen konzipiert. Damit sind die Anforderungen für die Strukturierung und die Funktionalität bekannt, so daß eine Entwicklung dieser Aspekte integriert erfolgen kann. Diese Vorgehensweise entspricht der Entwicklungsmethodik von informationsintensiven Websites. Sie bedingt jedoch eine weitestgehende Erfassung aller Szenarien der Anwendung.

Sichtenorientierte Entwicklung: Es wird ein Skelett oder eine Architektur der Anwendung entwickelt. Die einzelnen Sichten werden schrittweise und an ihren Schnittstellen integriert entwickelt. Darauf aufbauend können die Strukturierung, der Story-Raum und die Funktionalität entwickelt werden. Diese Vorgehensweise eignet sich besonders für gut strukturierte Anwendungsgebiete mit separierbaren Datenbeständen. Sie bedingt jedoch eine höhere Disziplin und Koordinierung bei der integrierten Entwicklung.

Schichtenbasierte Entwicklung: Es werden zuerst alle Aspekte auf der Motivationsschicht, danach auf der Geschäftsprozeßschicht, dann auf der Aktionsschicht und abschließend die Aspekte auf der konzeptionellen Schicht entwickelt. Nach Abschluß des konzeptionellen Entwurfes wird eine Transformation hin zur logischen Spezifikation vorgenommen. Dieser Zugang erfordert wenige Korrekturen im Entwicklungsprozeß und erscheint deshalb besonders geeignet. Er wird im weiteren präferiert.

Wir kombinieren diese Vorgehensmodelle zu einem schichtenbasierten Vorgehensmodell. Innerhalb einer Abstraktionsschicht determiniert der Interaktionsraum die anderen Aspekte.

Damit erhalten wir ein Vorgehensmodell, dessen Schrittfolge in Bild 17 dargestellt wird und das als Grundlage für die einzelnen Entwicklungsschritte dient.

Die einzelnen Schritte in Bild 17 sind die folgenden:

* Motivationsschicht

1. Entwicklung der Motivation und der Ziele der Anwendung, Informationsanalyse

2. Entwicklung des Lastenheftes zur Anwendung

* Geschäftsprozeßschicht

3. Separation der Systemes in Komponenten und Entwicklung der Architektur des Systemes

4. Skizzierung des Story-Raumes, Formulierung der Interaktivität für das Pflichtenheft

5. Skizzierung der Sichten-Suite für die einzelnen Komponenten, der Dienste und des Austauschrahmens, Formulierung der Verteilung und Strukturierung für das Pflichtenheft

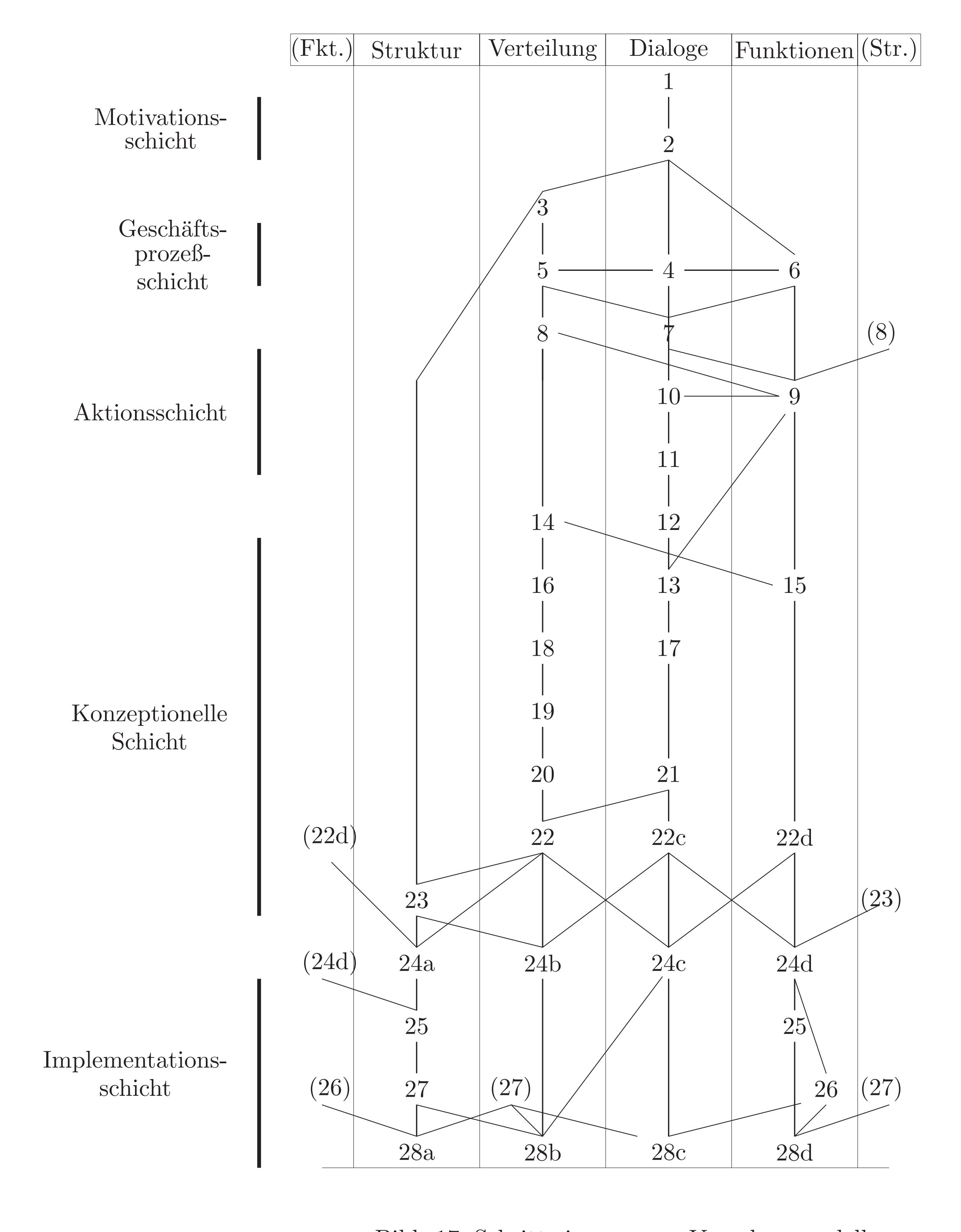


Bild 17: Schritte in unserem Vorgehensmodell

6. Spezifikation der Business-Prozesse, Formulierung der Funktionalität für das Pflichtenheft

* Aktionsschicht

7. Spezifikation der Szenario der Anwendung
8. Beschreibung der Haupttypen der einzelnen Sichten und deren Assoziationen
9. Entwicklung der Integritätsbedingungen und deren Erzwingungsstrategie
10. Spezifikation der Benutzeraktionen, Rollen, Skizzierung der Content-Typen
11. Spezifikation der Qualitätsanforderungen und deren Umsetzung im System, Entwicklung von Sicherungsstrategien

* Konzeptionelle Schicht

12. Spezifikation des Story-Raumes
13. Spezifikation der Akteure, ihrer Portfolio, Rollen, Rechte, Profile
14. Spezifikation der Sichten-Suite, der Dienste und Austauschrahmen
15. Entwicklung der Workflows
16. Kontrolle der Content-Typen anhand von Content-Objekten, Validierung der statischen Semantik, Kontrolle der Integritätserzwingung
17. Spezifikation der Szenen, der Dialogschritte, der Bedingungen für die Stories, der Handlungsübergänge
18. Spezifikation der Content-Typen-Suite, der notwendigen Funktionalität zu deren Unterstützung
19. Modulare Verfeinerung der Datentypen
20. Normalisierung der entwickelten Datentypen
21. Kontrolle des Story-Raumes anhand der Szenario, Ableitung weiterer möglicher Szenario, Blockierung unerwünschter Szenario, Ableitung der Verlinkungs- und Navigationsstruktur, Kontrolle der unterstützten Funktionalität
22. Spezifikation der Funktionalität, Kontrolle des Verhaltens der Anwendung, Abstimmung der Unterstützung für Dienste, Austauschrahmen, Kollaboration
23. Integration der Sichten-Suite anhand der Architektur des Systemes, Auflösung der Entwurfsobligationen

* Implementationsschicht

24. Transformation der konzeptionellen Modelle in logische Modelle zur Darstellung der Strukturierung, Funktionalität, Interaktivität und Verteilung
25. Restrukturierung und Optimierung auf der Grundlage von Performanzbetrachtungen und des Tuning
26. Ableitung des Dienstverwaltungssystemes, der Protokolle und der Funktionen zur Unterstützung der Verteilung
27. Transformation der logischen Modelle in physische Modelle des DBMS
28. Kontrolle der Dauerhaftigkeit und der Skalierbarkeit der Lösung, Entwicklung von Erweiterungs- und Migrationstrategien unter Berücksichtigung möglicher Technologieentwicklungen und Veränderungen in der Anwendung

4 Sprachen zur Darstellung der Strukturierung

> Was du ererbt von deinen Vätern hast,
> Erwirb es, um es zu besitzen.
> Was man nicht nützt, ist eine schwere Last;
> Nur was der Augenblick erschafft, das kann man nützen.
> *Goethe, Faust, Erster Teil, Nacht, Faust*

Eine Sprache zur Beschreibung der Strukturierung von Datenbank-Anwendungen verfügt über Konstrukte zur Darstellung der Struktur einer Anwendung. Falls diese Sprache nicht-zyklisch und induktiv aufgebaut ist, ist damit auch eine Einbettung in die Sprache der Prädikatenlogik (der ersten Stufe) gegeben. Deshalb lassen sich dann statische Integritätsbedingungen als Formeln der Prädikatenlogik mit einer Standardinterpretation angeben. Mit der Sprachkonstruktion und mit Annahmen aus dem Umfeld werden *implizite Integritätsbedingungen* aufgenommen. Die Sprache zur Beschreibung der Strukturierung von Datenbanksystemen wird genutzt, um diese mit einem sogenannten *Datenbank-Schema* zu beschreiben. *Inhalte* eines statischen Modelles sind daher:

Strukturen einer Anwendung,

Statische Integritätsbedingungen einer Anwendung (meist für die zusätzliche Beschränkung evt. in einer Anwendung vorkommender Daten) und

Common-sense-Annahmen (über das Modell, die Modellierungsart, über die Interpretation der Daten etc.).

Damit wird das *Wissen über die statischen Gesichtspunkte einer Anwendung* modelliert durch:

Die ***Spezifikation der Struktur*** in Abhängigkeit vom Typensystem mit der Spezifikation des *Seienden* (entity), der *Beziehungen* (relationship) und der *Eigenschaften* (Attribute).
Dinge stehen in Beziehung bzw. besitzen Eigenschaften, die klassifiziert werden durch eine *Rolle* oder durch *Klassenbildung*.
Die Gesamtheit der Dinge wird unter Berücksichtigung der Beziehungen untereinander modelliert:

- *Aussonderung* (Separation/Spezialisierung),
- *Verallgemeinerung* (Generalisierung von Gemeinsamkeiten) und
- *Aggregation* (zur Darstellung komplexerer Daten mit entsprechenden Operationen).

Die ***Spezifikation der statischen Semantik,*** d.h. durch einschränkende Bedingungen für wirklichkeitsgetreue Nachbildung der Anwendung wie

- die eindeutige Bestimmung aller Objekte durch *Schlüsselbedingungen*,
- die Hierarchie der Objekte (*Aussonderungsbedingungen* (specialization, IsA), *Verallgemeinerungsbedingungen* (partition constraints, uniqueness constraints))
- und Bedingungen für Beziehungsklassen wie die folgenden:
  - Darstellung eines funktionalen Zusammenhangs (*viele-eins-Bedingung*),
  - Bedingungen zur Assoziation mit Komponentenobjekten (*Seinsbedingung* (existence constraint)) und
  - *Verweisbedingungen* auf Objekte der Komponentenklassen,

  sowie
- allgemeine Bedingungen (*inhärente Bedingungen des Modells*) wie die folgenden:
  - *Gesamtheitsregel* (universe of discourse)
  - *Verneinungsregel*

Sichten und abgeleitete Begriffe sind erschließbare Objekte und werden durch Anwendung von Spezifikationen aus den Objekten der Datenbank erzeugt.

Das *allgemeine Vorgehen* der statischen Datenbankmodellierungssprachen läßt sich somit wie folgt charakterisieren:

- Typen sind über ihre Typausdrücke definiert. Den (freien) Variablen werden wiederum Typen zugeordnet.
  - Die Zuordnungsvorschrift für Typausdrücke kann sowohl hierarchisch als auch zyklisch sein. Wählt man eine zyklische Struktur, dann sind meist nur `Topoi-Semantiken` geeignet. Wählt man hierarchische Strukturen, dann kann meist eine `Mengensemantik` noch garantiert werden.
  - Typen haben eine assoziierte statische Semantik.
  - Typen haben Operationen zu ihrer Manipulation und Veränderung. Man kann diese Operationen `generisch` definieren, wenn die Typenstruktur hierarchisch aufgebaut ist. Einige Operationen können auch Prädikate sein.
- Klassen sind Typen zugeordnet.
  - Sie stellen "Container" für die Objekte des jeweiligen Typs dar.
  - Die assoziierte statische Semantik der Typen muß zu jedem Zeitpunkt für eine Klasse erfüllt sein.
  - Die Operationen der Typen werden auf Klassen ausgeführt.

  Wir bezeichnen Typen mit ihrem Namen, z.B. T und die zugehörigen Klassen mit einer Annotation zum Typennamen, z.B. T^C (C steht für Klasse).

Es sind verschiedene Modelle möglich. Jedes Modell ist durch eine Menge von inhärenten Bedingungen gekennzeichnet. Jeder benutzte Typ hat neben *Konstruktor, Selektoren* (für Retrieval) und *Updatefunktionen*, *Korrektheitskriterien*, *default-Regeln* auch eine *Benutzerrepräsentation* und eine *physische Repräsentation*.

Günstig ist eine *graphische* Repräsentation.

Eines der populärsten Modelle ist das **Entity-Relationship-Modell**. Wir erweitern dieses Modell zu einem *Higher-Order Entity-Relationship-Modell* (HERM). Es umfaßt die folgenden Konstrukte:

- Attribut-Typen können einfache oder auf der Grundlage von Konstruktoren wie Mengenkonstruktor, Tupelkonstruktor, Listenkonstruktor, Multimengenkonstruktor induktiv konstruierte komplexe Attribut-Typen sein. Sie werden induktiv definiert:

  Basis-Datentypen sind parametrisierte Typen $T = (dom(T), ops(T), pred(T))$ des DBMS. Sie sind gegeben durch eine Bezeichnung T (evt. auch mit Abkürzung), einen Wertebereich $dom(T)$, eine Menge von Funktionen $ops(T)$ und eine Menge $pred(T)$ von Prädikaten.

  Oft wird auch der Basis-Datentyp mit einem Informationstyp assoziiert.
  Ein Beispiel ist der Typ der ganzen Zahlen in der 4-Byte-Repräsentation
  integer := *(IntegerSet*$_{4Byte}$, $\{0, s, +, -, *, \div, \}$, $\{=, \leq\}$*) mit der Nachfolgefunktion s* .
  Basis-Datentypen verfügen neben dem Wertebereich auch über Funktionen und Prädikate. Sie sind außerdem durch eine Reihe von Eigenschaften eingeschränkt, die im Datenbanksystem zu beachten sind und oft im Entwurf übersehen werden:

  - Die *Präzision und Genauigkeit* sind ggf. für Typen wie *REAL* eingeschränkt.
  - Die *Granularität* von Daten kann sehr unterschiedlich sein. Die *Skalierung* von Datentypen kann sich ggf. auch auf die Funktionalität auswirken.
  - Datentypen verfügen nur ggf. über eine eigene *Ordnungsbeziehung*.
  - Datentypen verfügen ggf. über eine *Klassifikation* innerhalb der Daten des Wertebereiches. Diese Klassifikation kann einfach oder mehrfach hierarchisch, analytisch oder synthetisch, monothetisch oder polythetisch und ein- oder mehrdimensional sein.

- Datentypen können über unterschiedliche *Präsentationsformen* verfügen. Das *Format* umfaßt Länge und Größe.
- Datentypen können auf unterschiedliche Art *abgespeichert* werden.
- Datentypen verfügen über eigenständige *Default-* und *Nullwerte*.
- Datentypen können durch *Casting-Funktionen* aufeinander abgebildet werden.
- Datentypen sind bestimmten *Anwendungen* und *Arbeitsgebieten zugeordnet*.
- Die Funktionen und Prädikate lassen unterschiedliche *Berechnungen* zu, die sich auf die Erfassung, Berechnung, Algorithmen etc. auswirken.
- Bestimmte Funktionen, wie z.B. der Durchschnitt, sind evt. *anders* oder *gar nicht definiert*.
- Datentypen sind oft mit *Maßeinheiten* ausgewiesen, womit auch Berechnungen unterlegt werden müssen.

Basis-Datentypen sind meist auch in einem *Typenverband* geordnet.
Neben den Basis-Datentypen des DBMS kann auch eine Anwendung über eigene Basis-Datentypen verfügen. Wir können z.B. den Typ *varnumbersequence20* zur Darstellung von Telefonnummern mit einer angepaßten Ordnungsbeziehung und ohne Unterdrückung führender Nullen einführen. Analog kann ein Typ *EmailTyp* oder *URL* eingeführt werden.

Attribut-Typen werden über einem Basis-Datentypen-System und einem Markierungssystem L für Attributnamen induktiv ausschließlich durch die folgenden beiden Regeln definiert:

- Ein Attribut-Typ ist für eine Markierung A und einen Basis-Datentyp durch einen Ausdruck $A :: T$ gegeben. Der Wertebereich $dom(A)$ des Attribut-Typs ist der Wertebereich des Basis-Datentyps. Der Wertebereich des leeren Datentyps λ besteht aus $\perp$.
- Sind $X_1, ..., X_n, Y$ Attribut-Typen und A, B, C, D Markierungen, dann sind $A(X_1, ..., X_n)$ (Tupel- oder Produkt-Konstruktor), $A\{Y\}$ (Mengen-Konstruktor), $A < Y >$ (Listenkonstruktur), $A[Y]$ (Konstruktor für optionale Elemente), $A\{\!|Y|\!\}$ (Konstruktor für Multimengen).
  Die entsprechenden Wertebereiche sind durch Anwendung der Konstruktion gegeben, z.B.
  $dom(A(X_1, ..., X_n)) = dom(X_1) \times ... \times dom(X_n)$ und $dom(A\{Y\}) = 2^{dom(Y)}$.
  Markierungen können auch weggelassen werden.

Beispiele von komplexeren Attributen sind
Name *(Vornamen<(Vorname :: varstring15 , Benutzung :: string1)>,*
Familienname :: varstring30, [Geburtsname :: varstring30,]
[Titel:$\{$*AkademischeTitel :: varstring10* $\} \dot\cup$ *FamilienTitel :: varstring10])*
Kontakt *(Tel(*$\{$*dienstl :: varnumbersequence20* $\}$*, privat :: varnumbersequence20),*
email :: emailType, ...)
Geburtsdatum :: date .

Attribute können in einer verkürzten Notation verwendet werden, wenn dies eindeutig im Schema bleibt. Das Attribut *Kontakt* ist z.B. dann auch ohne seine Bestandteile verwendbar.

Attribute sind hierarchisch strukturiert wie - im Falle des *Namens* einer *Person* - der Baum in Bild 18 zeigt. Diese hierarchische Struktur ermöglicht auch Elemente auszuzeichnen, z.B. mit der Eigenschaft Element eines Schlüssels zu sein. So kann z.B. zum Schlüssel das Teilattribut

Name *(Vornamen, Familienname, [Geburtsname])*

hinzugenommen werden, wobei wir als Abkürzungsregel benutzen, daß mit dem Nennen eines Bezeichners auch der damit verbundene Teilbaum mit übernommen wird, z.B. für *Vornamen* auch die gesamte Teilstruktur *Vornamen<(Vorname :: varstring15 , Benutzung :: string1)>* .

- HERM-Typen werden induktiv aufeinander basierend definiert.

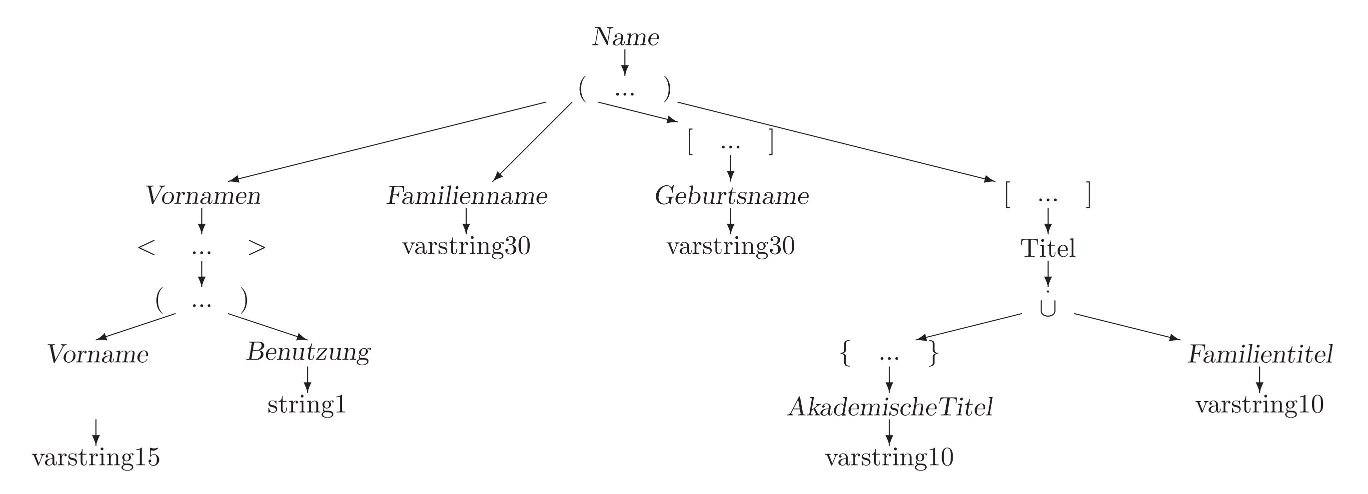


Bild 18: Semi-strukturiertes Attribut *Name*

Entity-Typ: Eine Seiendenklasse (Objektklasse) (Entity-Klasse im weiteren) wird durch einen Entity-Typ dargestellt. Ein Entity-Typ besteht aus einer nichtleeren Folge von Attributen und einer Menge von statischen Integritätsbedingungen. Der Primärschlüssel wird direkt durch Unterstreichen der Attribute angegeben. Ist die Menge der statischen Integritätsbedingungen leer, dann kann sie auch weggelassen werden. Eine Klasse von der Struktur des Entity-Typs ist gültig, falls alle Integritätsbedingungen gelten. Wir folgen der klassischen Notation, bei der ein Entity-Typ mit einer Definitionsgleichung dargestellt wird. Zum Beispiel ist ein Person-Typ spezifiziert durch

Person = (Name, Adresse, Kontakt, GebDatum, <u>PersNr</u> : StudNr ∪̇ MitarNr, ..., ∅)
mit einer Folge von Attributen. Markierungen sind als solche ausgewiesen.

Ein Entity-Typ wird durch ein Rechteck graphisch repräsentiert.

Eine Entity-Klasse besteht aus einer Menge von Objekten vom Entity-Typ, die die statischen Integritätsbedingungen des Entity-Typen erfüllt.

Zum Beispiel ist das folgende Objekt mit dem Identifikator *β*
β: ((<(Karl,z),(Bernhard,r)>, Thalheim, {Prof., Dr.rer.nat.habil., Dipl.-Math.}),
BTU Cottbus, (({ +49 355 692700, +49 355 692397}, +49 355 824054),
thalheim@informatik.tu-cottbus.de), 10.3.52, 637861)
vom Entity-Typ *Person*, wobei mit 'z' der Zusatzname und mit 'r' der Rufname bezeichnet wird.

Einfacher Relationship-Typ: Ein Relationship-Typ (erster Ordnung) besteht aus einer nicht-leeren Folge von Entity-Typen, einer Menge von Attributen und einer Menge von statischen Integritätsbedingungen. Eine Menge von der Struktur des Relationship-Typen ist eine gültige Menge, wenn sie den statischen Integritätsbedingungen genügt. Elemente können markiert sein.

Ein Beispiel sind die Relationship-Typen
InGruppe = (Person, Gruppe, { Zeit(Von [,Bis]), Funktion }, ∅)
DirektVoraussetz = (setztVoraus: Kurs, vorausges : Kurs, ∅, ∅)
Professor = (Person, { Berufungsgebiet }, ∅) .

Ein Relationship-Typ wird mit einer Raute graphisch repräsentiert. Wir erlauben auch optionale Komponenten von Relationship-Typen, solange eine Identifikation über die obligatorischen Elemente definiert ist.

Ein Objekt eines Relationship-Typs ist ein Tupel, das zu den jeweiligen Elementen auf die entsprechenden Objekte der Klasse der Elemente durch Angabe von identifizierenden Werten (Identifikator bzw. Primärschlüssel bzw. anderer Schlüssel) verweist und Werte für die Attribute des Relationship-Typs besitzt.

Eine Relationship-Klasse besteht aus Objekten des Relationship-Typs, die den statischen

Integritätsbedingungen genügen.

z.B. sind Objekte der Typen *Professor*, *InGruppe* und *DirektVoraussetz*

Profβ: (637861, Datenbank- und Informationssysteme)
Senator3β: (637861, Senat, (1995,1998), Dekan)
Senator5β: (637861, Senat, (2000), Vorsitzender)
VorausDBIVHaupt: (DBIV, DBI) .

Cluster-Typ Eine *disjunkte* Vereinigung von bereits konstruierten Typen wird als Cluster-Typ bezeichnet. Ein Cluster-Typ wird mit einem $\bigoplus$-Zeichen graphisch repräsentiert.

Beispiele sind durch folgende Typen gegeben:

$JuristischePerson = Person \dot{\cup} Betrieb \dot{\cup} Vereinigung$

$Gruppe = Senat \dot{\cup} Arbeitsgruppe \dot{\cup} Vereinigung$,

die den Typ *JuristischePerson* bzw. *Gruppe* als disjunkte Vereinigung von anderen Typen einführen.

Cluster-Typen können weitere Attribute besitzen. In diesem Fall wird der Cluster-Typ durch eine Raute mit den Attributen repräsentiert.

Objekte von Cluster-Typen sind analog zu den Objekten anderer Typen durch entsprechende Zuordnung zu den Element-Typen eingeführt. So können z.B. die Objekte *β, LIM, CottbusNet e.V.* juristische Personen sein.

Relationship-Typ höherer Ordnung: Ein Relationship-Typ i-ter Ordnung besteht aus einer nichtleeren Folge von Entity- und Relationship-Typen einer Ordnung von maximal (i-1), wobei ein Typ (i-1)-ter Ordnung sein muß, einer Menge von Attributen und einer Menge von statischen Integritätsbedingungen. Eine Menge von der Struktur des Relationship-Typen ist eine gültige Menge, wenn sie den statischen Integritätsbedingungen genügt. Eine Identifikation kann sowohl aus den Elementen bestehen als auch aus den Attributen.

Es ist mitunter vorteilhaft, über Relationship-Typen höherer Ordnung zu verfügen, wie Bild 19 zeigt. Im oberen Diagramm muß eine zusätzliche Integritätsbedingung zwischen den Typen *eingeschriebenIn* und *Vorlesung* gelten, weil man sich nur dann einschreiben kann, wenn diese Vorlesung existiert.

Ein etwas komplexeres Beispiel ist das Beispiel in Bild 20. Eine Lehrveranstaltung, z.B. eine *Vorlesung*, wird durch einen *Lehrstuhl* angeboten. Dieses Angebot kann angenommen werden. Dann wird die Lehrveranstaltung geplant. Wird sie auch gehalten, dann werden die aktuellen Daten in der Klasse zum Typ *GehalteneLehrveranst* gespeichert. Der Typus und die Raumzuordnung können sich vom Vorschlag zum Plan und für den Raum vom Plan zu den gehaltenen Lehrveranstaltungen ändern. Ein Vorschlag für eine Lehrveranstaltung wird durch Berechtigte eingetragen. Eine Person ist für die Lehrveranstaltung verantwortlich. Eine Lehrveranstaltung kann für mehrere Studiengänge angeboten werden.

Wir wollen hier nicht die vollständige Entfaltung von Objekten zu Typen höherer Ordnung fordern. Deshalb erbt ein Relationship-Typ höherer Ordnung nur die Identifikation seiner Elemente oder - wenn wir an einer vollständigen Wertedarstellung interessiert sind - nur die identifizierenden Werte der Objekte seiner Komponenten. So können z.B. Objekte vom Typ *geplanteLehrveranstaltung* in Bild 20 auch nur auf Objekte verweisen, die *Kurs, Semester, Professor* bezeichnen, wenn wir voraussetzen, daß ein Schlüssel des Typs *angeboteneVorlesung* aus *Kurs, Semester, Professor* besteht.

Ein Objekt vom Typ

angeboteneVorlesung = (Kurs, Semester, Studiengänge,
Professor, eingetragen, Verantwortlicher4LV, Raumwunsch, Typus, { *Zeit* }, $\emptyset$)

ist z.B.

VorlesungDBIVSS02: (DBIV, SS2002, { Informatik, IMT },
637861, KK, 637861, SR1, Vorlesung/Übung/Praktikum 2+2+2, Mo. 1.DS) .

Generalisierung versus Spezialisierung: Ein Cluster-Typ erlaubt die explizite Darstellung einer Generalisierung. Ein unärer Relationship-Typ stellt dagegen eine Spezialisierung dar, wenn

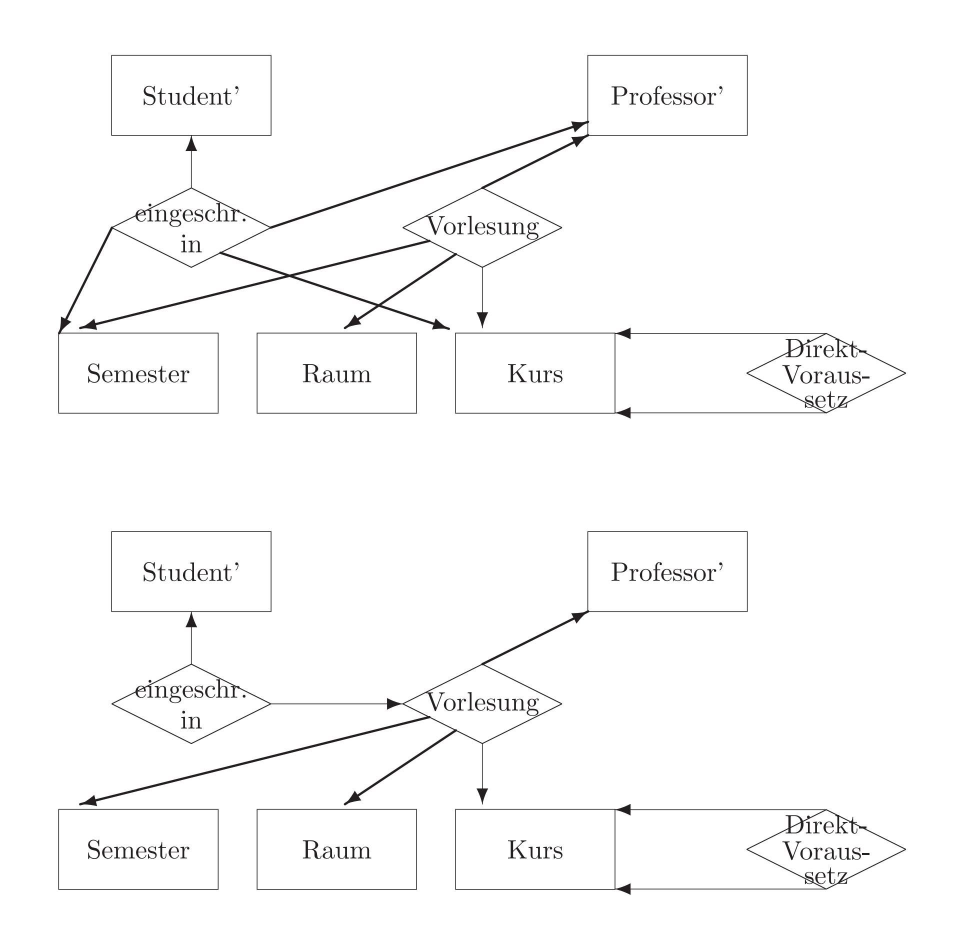


Bild 19: HERM Diagramme mit und ohne Relationship-Typen höherer Ordnung

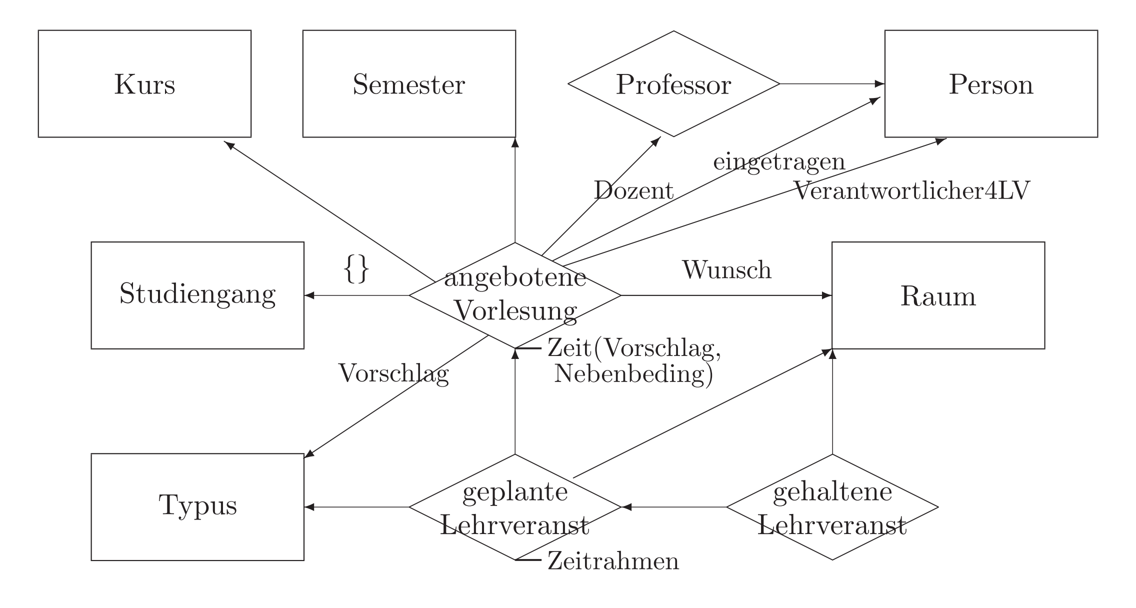


Bild 20: HERM Diagramm zu unserem Hauptbeispiel

der Relationship-Typ bzw. Entity-Typ als sein Element diesen identifiziert. Rollen werden oft durch einen generischen Typ mit der Bezeichnung *IsA* dargestellt. Da die relationalen Schemata auch ohne diesen Typ auskommen, bevorzugen wir die Darstellung als Rolle mit unären Relationship-Typen oder ggf. auch mehrstelligen Relationship-Typen, falls die Rolle durch eine Beziehung zu anderen Typen ausgezeichnet ist. Damit sind wir in der Lage, zwischen Generalisierung und Spezialisierung zu unterscheiden.

Rollen, die exklusiv bzw. hierarchisch sind, lassen sich auch anstelle einer HERM-Rautenstruktur durch hierarchische Strukturen abbilden, wie in Bild 21 dargestellt. Welche Darstellungsform gewählt wird, hängt vom erforderlichen Detaillierungsgrad ab. Sollen Attribute mit dargestellt werden, wird das hierarchische ER-Modell sehr schnell zu unübersichtlich. In den ersten Abstraktionsschichten stellt es aber eine gute Alternative zum HERM-Diagramm zum.

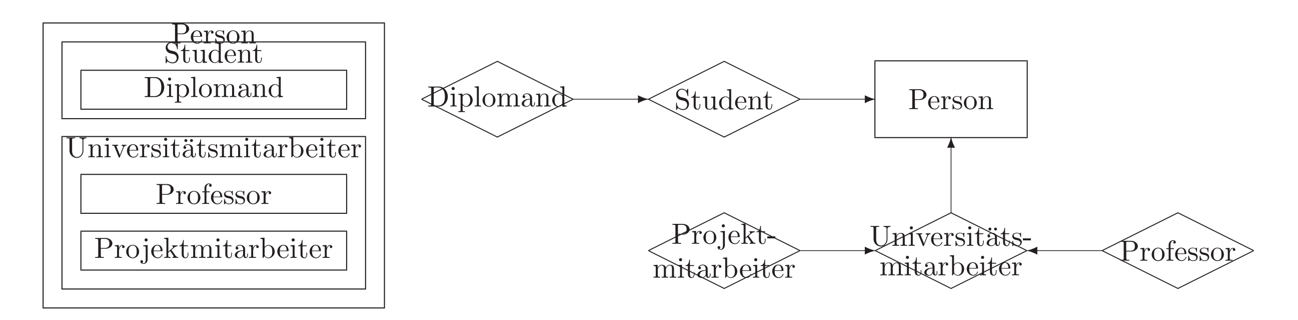


Bild 21: Hierarchisches ER-Diagramm versus HERM Diagramm

Aggregation: Wir können die Konstruktion von Relationship-Typen zu einer allgemeinen *Aggregationskonstruktion* erweitern, indem wir weitere Konstruktoren zulassen:

- Vereinigung,
- Mengenbildung,
- Aggregation durch Beziehungsklasse und
- Abstraktion durch Komponentenbildung.

Klassen werden mit der hochgestellten Annotation 'C' und dem Typnamen bezeichnet. Z.B. sind $Person^C$ und $InGruppe^C$ Klassen entsprechenden Typs.

- Statische Integritätsbedingungen: Die Semantikspezifikationssprache umfaßt Schlüssel und Integritätsbedingungen, wie funktionale Abhängigkeiten, Exklusions- und Inklusionsabhängigkeiten, mehrwertige Abhängigkeiten, Viele-Eins-Bedingungen, Seinsbedingungen (Existenzbeziehung), Verweisbedingungen, Teiltypenbedingungen und Regeln, wie z.B. die Gesamtheitsregel, die Verneinungsregel und die Sichtregeln, sowie vor allem Komplexitätsbedingungen (Kardinalitätbedingungen) zur Spezifikation der Beziehung zwischen einem Relationship-Typen und seinen Komponenten.

Statische Integritätsbedingungen werden als Formeln der hierarchischen Prädikatenlogik allgemein dargestellt. Wir verwenden jedoch die üblichen Kurzdarstellungen.
Wir gehen davon aus, daß statische Integritätsbedingungen einer Interpretation mit einer "Normallogik" unterliegen. Mitunter wird auch im Entwurf eine Integritätsbedingung mit einer schwachen, deontischen Interpretation benutzt, bei der ihre Gültigkeit für die meisten Objekte einer Datenbank oder einer Klasse gefordert wird. Mitunter wird auch eine strikte Form der Interpretation genutzt, bei der z.B. obere bzw. untere Schranken für Kardinalitätsbeschränkungen auch durch entsprechende Objektmengen genau erfüllt sein müssen.

Wir verwenden im weiteren folgende Klassen von Integritätsbedingungen:

Schlüssel dienen der Darstellung der Identifizierbarkeit von Objektmengen, insbesondere in Entity-Klassen). Wir nehmen an, daß Entity-Klassen stets eigen identifiziert sind, d.h.

Mengen sind. Eine Teilmenge der Strukturelemente kann auch als Schlüssel dienen. Gewöhnlich hat jeder Typ mehr als einen Schlüssel. Deshalb verwenden wir von vornherein Schlüsselmengen. Der Primärschlüssel eines Entity-Typs wird direkt angegeben und kann in der Schlüsselmenge weggelassen werden.

Wir nehmen z.B. für das Diagramm in Bild 20 folgende Schlüssel an:
Key(Person) = { { *PersNr* }, { *Name, Geburtsdatum* } }

Relationship-Typen haben ggf. auch eigene Attribute, die auch Bestandteile eines Schlüssels sind.

Zum Beispiel nehmen wir für das obige Beispiel an, daß die Zeit essentiell für *InGruppe* ist, d.h.
Key(InGruppe) = {{ *Person, Gruppe, Zeit* }} oder
Key'(InGruppe) = {{ *Person, Gruppe, Zeit, Funktion* }}

Weiterhin kann z.B. gelten
Key(Vorlesung) = { {*Kurs, Semester*}, {*Semester, Raum, Zeit*}, {*Semester, Dozent, Zeit*} }

Schlüssel folgen der Komponentenkonstruktion und können auch für einen Teil gelten, z.B. *Name(Vornamen<(Vorname,use)>, FamName).*
Mindestens ein Schlüssel wird über die Komponente an den Relationship-Typen 'vererbt'.

Funktionale Abhängigkeiten sind eine wichtige Gruppe von Abhängigkeiten. Eine funktionale Abhängigkeit $R : X \to Y$ ist für einen Typ R und Teilmengen X, Y seiner Elemente definiert. Sie gilt in einer Klasse R^C, falls die Gleichheit von Objekten o, o' aus R^C über X die Gleichheit über Y für o, o' impliziert.

Funktionale Beziehungen von Attributgruppen in unserem Beispiel sind
geplanteLV : {*Semester, Zeitrahmen, Raum*} → {{*Studiengang*}, *Professor, Kurs*}
geplanteLV : {*Professor, Semester, Zeitrahmen*} → {*Kurs, Raum*}
angeboteneLV: {*Semester, Kurs*} → {*Professor*} .

Kardinalitätsbeschränkungen werden als kombinatorische Beschränkungen in der (min,max)-Notation und der Partizipations-Semantik als Paar von Kardinalitäten verwendet. Damit unterscheidet sich unsere Notation von der Lookup-Semantik, die z.B. UML verwendet. Die letztere kann jedoch in einer n..m-Notation ebenso mitgeführt werden. Wir betrachten hierzu ein vereinfachtes Diagramm in Bild 22.

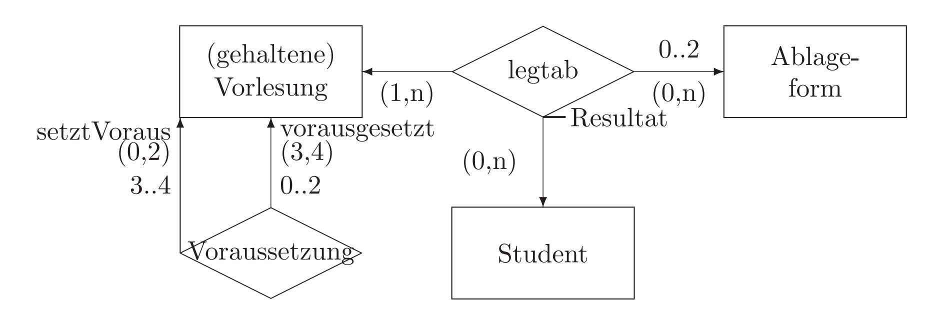


Bild 22: Kardinalitätsbeschränkungen im Vorlesungsbeispiel

Eine Kardinalitätsbeschränkung $card(R, R_i) = (n, m)$ gilt in einer Klasse R^C, falls jedes Objekt o_i von R_i^C in R^C mindestens n-mal und höchstens m-mal vorkommt.

Eine Kardinalitätsbeschränkung in der Lookup-Notation $look(R, R_i) = (n, m)$ gilt in einer Klasse R^C mit k Elementen, falls zu jeder Kombination von Objekten o_j von R_j^C $(j \neq i, 1 \leq j \leq k)$ mindestens n und höchstens m entsprechende Objekte o_i aus R_i^C in der Klasse R^C vorkommen.

Im Fall binärer Relationship-Typen kann man damit einem Objekt o von R_i mindestens n und höchstens m Objekte aus R_j^C zuordnen, d.h. das Objekt sieht vermittels R^C höchstens m und mindestens n Objekte aus der anderen Klasse.

Die Lookup-Notation ist für binäre Relationship-Typen *ohne eigene Attribute* äquivalent zur Partizipation-Notation. Sie wird jedoch am anderen Element angetragen. Im Beispiel nehmen an, daß

card(Voraussetzung, setztVoraus) = (0,2)
look(Voraussetzung, setztVoraus) = 3..4
card(Voraussetzung, vorausgesetzt) = (3,4)
look(Voraussetzung, vorausgesetzt) = 0..2

gilt. Damit haben wir äquivalente Formen.

Für n-äre Relationship-Typen *ohne eigene Attribute* ist die Lookup-Notation $look(R, R_i) = n..m$ äquivalent zur verallgemeinerten Kardinalitätsabhängigkeit $card(R, R \setminus R_i) = (n, m)$.

In unserem Beispiel gilt z.B. die Einschränkung, daß erst dann ein Eintrag in die Klasse *legtab* geführt wird, wenn der Student eine Vorlesung erfolgreich abgelegt hat.

Die Lookup-Bedingung *look(legtab, Ablageform) = 0..2* stellt dar, daß nur *Prüfung* und *Schein* bzw. *Schein* und *Praktikum* bzw. *Prüfung* und *Praktikum* absolviert werden müssen. Diese Bedingung ist äquivalent zu

card(legtab, Student Vorlesung) = (0,2).

Eine Kardinalitätsbeschränkung $card(R, R_i) = (0, 1)$ ist äquivalent zur funktionalen Abhängigkeit $R : \{R_i\} \to R$.

Eine Lookup-Kardinalitätsbeschränkung $look(R, R_i) = 0..1$ ist äquivalent zur funktionalen Abhängigkeit $R : R \setminus \{R_i\} \to R$.

Weiterhin können wir z.B. fordern, daß nur solche Vorlesungen als gehalten gelten, die auch zu studentischer Beteiligung geführt haben. Dies wird durch *card(legtab, Vorlesung) = (1,n)* dargestellt.

Eine strengere Bedingung ist, daß dies auch für das Semester gelten muß. Dann können wir spezifizieren

look(legtab, Student) = 1..n bzw. *card(legtab, Vorlesung Semester) = (1,n).*

Für Relationship-Typen mit eigenen Attributen ist die Lookup-Notation in verschiedenen Formen definiert.

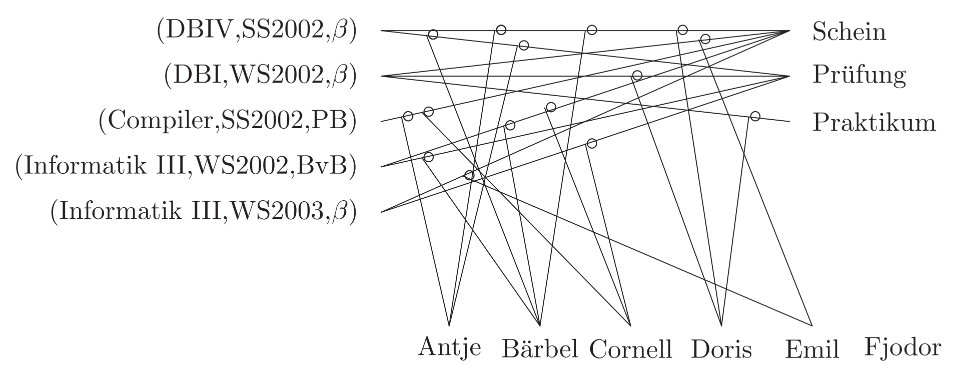


Bild 23: Beziehungen der Objekte im Vorlesungsbeispiel

Wir betrachten in diesem Beispiel in Bild 23 eine kleine Klasse mit 14 Objekten. Z.B. hat *Bärbel* sowohl die *(Informatik III,WS2002,BvB)* als auch *(DBIV,SS2002,β)* mit *Prüfung* und *Schein* abgelegt, *Emil* dagegen nur *Scheine* in *(Informatik III,WS2002,BvB)* und *(DBI,WS2002,β)*.

Kardinalitätsbeschränkungen sind mitunter nicht erfüllbar in nicht-leeren, endlichen Klassen. Ein Beispiel einer solchen nicht-erfüllbaren Menge von Integritätsbedingungen ist das Paar

card(Voraussetzung, setztVoraus) = (0,2)
card(Voraussetzung, vorausgesetzt) = (3,4) .

Dagegen ist

card(Voraussetzung, setztVoraus) = (0,3)

card(Voraussetzung, vorausgesetzt) = (3,4)
erfüllbar und impliziert
card(Voraussetzung, setztVoraus) = (3,3)
card(Voraussetzung, vorausgesetzt) = (3,3) .

Mehrwertige Abhängigkeiten stellen im Entwurf i.a. die Separation von Gesichtpunkten bzw. Aspekten dar. Sie werden oft weggelassen, da ihre mathematische Notation schwierig nachzuvollziehen ist.

Eine mehrwertige Abhängigkeit $X \twoheadrightarrow Y|Z$ wird für einen Typ $R = (U_R, \Sigma_R)$, mit Teilmengen $X, Y \subseteq U_R$ und $Z = U_R \setminus (Y \cup X)$ definiert und gilt in einer Klasse Relation R^C über R (dargestellt durch $R^C \models X \rightarrow\rightarrow Y|Z$), falls für alle $o, o' \in R^C$, die den gleichen Wert für die X-Elemente von R haben, ein Objekt o'' in R^C existiert, das aus der Faltung von o und o' hervorgehen kann, d.h. formal

für alle $o, o' \in R^C$ mit $o =_X o'$ existiert ein Objekt $o'' \in R^C$ mit $o'' =_{X \cup Y} o$ und $o'' =_{X \cup Z} o'$.

Eine nützliche, allgemein bekannte Eigenschaft von mehrwertigen Abhängigkeiten ist die Dekompositionseigenschaft. Es gilt $R^C \models X \rightarrow\rightarrow Y|Z$ genau dann, wenn sich R^C nach $X \cup Y$ und $X \cup Z$ vertikal dekomponieren läßt, d.h. formal $R^C = R^C[X \cup Y] \bowtie R^C[X \cup Z]$.

Weniger bekannt ist dagegen, daß die Gültigkeit der mehrwertigen Abhängigkeit zu einem neuen äquivalenten Schema führt, bei dem der Typ R durch die dekomponierten Typen wie in Bild 24 ersetzt wird.

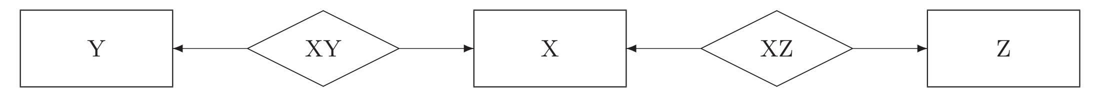


Bild 24: Die Zerlegung von R in zwei Relationship-Typen

Weitere relationale Integritätsbedingungen, z.B. Wertebereichsabhängigkeiten, können im erweiterten ER-Modell verwendet werden. So gilt in unserem Beispiel

Semester.Bezeichnung
$\in \{WS, SS\} \times \{x/x{+}1 | x \in 80..99, 00, 01, 02, ..., 17\}$.

Andere wichtige Klassen von Abhängigkeiten sind Exklusions- und Inklusionsabhängigkeiten.

- Ein Datenbank-Schema $\mathcal{ER}$ besteht aus einer Menge von Typen $\{T_i = (U_{T_i}, \Sigma_{T_i})\}$ und globalen statischen Integritätsbedingungen $\Sigma_{\mathcal{ER}}$.

Unsere Strukturierungssprache unterstützt das Abstraktionsschichtenmodell. Es kann die Strukturierung der Daten in jeder Schicht durch das Entity-Relationship-Modell repräsentiert werden. Wir verwenden dazu Schemata unterschiedlicher Abstraktheit und Granularität.

Datenstrukturierung des Lastenhefts: Es wird ein allgemeines HERM-Diagramm mit den Haupttypen entwickelt.

Datenstrukturierung des Pflichtenhefts: Es wird ein grobes HERM-Diagramm mit entsprechenden Integritätsbedingungen angegeben, das die Typen des Lastenhefts verfeinert. Die Verfeinerung findet durch Spezialisierung der Typen, Dekomposition, strukturelle Erweiterung, semantische Einschränkung, Separation von Aspekten und durch Instantiierung statt. Zusätzlich werden weitere Typen eingeführt.

Anwendungsschema: Das Anwendungsschema repräsentiert alle Typen, die für den Anwender eine Bedeutung haben. Die Typen stellen eine Verfeinerung der Typen des Pflichtenhefts dar oder sind neu eingeführt.

Konzeptionelles ER-Schema: Auf der konzeptionellen Schicht wird ein detailliertes HERM-Diagramm erstellt, das u.a. auch für alle Typen des Anwendungsschemas entsprechende Verfeinerungen enthält. Diese Beziehungen finden auch Eingang in die Sichten-Suite.

Logisches Schema: Das HERM-Schema wird in ein entsprechendes Schema des logischen Datenbank-Modelles transformiert. Es kann üblicherweise ein objekt-relationales oder relationales Schema, aber auch eine Beschreibung als XML-Schema oder DTD-Datei (document type definition) sein.

Diese Schemata sind aufeinander abbildbar. Demzufolge kann jede Entwurfseinheit einer höheren Schicht - so wie in Bild 14 auf Seite 33 dargestellt - einer Menge von Entwurfseinheiten der folgenden Schicht direkt zugeordnet werden.

Wir merken an, daß wir über zwei unterschiedliche Methoden zur Darstellung, Repräsentation, Verarbeitung und Speicherung von Objekten verfügen:

Klassen-Separation: Die Menge aller Objekte wird durch ein ER-Schema dargestellt. Jedes Objekt wird genau einer Klasse zugeordnet und in beliebig vielen anderen Klassen auf der Grundlage des ER-Schemas verwendet. Die Verwendung kann über einen Surrogat-Schlüssel, eine Markierung oder Werte zum ausgewählten Schlüssel des Objektes erfolgen.

Wir nennen diese Form der Behandlung von Objektmengen klassen-separierte Darstellung. Ein Objekt ist dann mit dem erweiterten ER-Modell als Schneeflocke mit einer Wurzel darstellbar.

Objekt-Entfaltung: Die Menge aller Objekte bildet unter Einbeziehung der Beziehungen der Objekte untereinander einen Objektmengen-Graphen. Wir können über diesem Graphen beliebige Überdeckungen $\mathcal{U}$ bilden, d.h. Mengen von Teilgraphen, die zusammen den Objektmengen-Graphen ergeben. Ein Teilgraph besitzt evt. ein Wurzel-Objekt, d.h. es gibt ein Objekt, das rekursiv auf alle anderen Objekte des Teilgraphen verweist. Besitzt jeder dieser Teilgraphen ein Wurzelobjekt, dann heißt $\mathcal{U}$ Objekt-Gesellschaft.

Damit ist in Objekt-Gesellschaften jedes Objekt ein *volles Objekt mit allen Eigenschaften*.

Ein Beispiel für eine Objekt-Entfaltung zum Schema in Bild 20 ist folgendes XML-Dokument:

```
<Lehrveranstaltung ID="120201" Titel = "DB-Programmierung" Typus = "TypusID1"
                                                    Erfuellung = "gehalten">
  <geplant durch="Fak.-Ref. Schenk"><Dozent Name = "beta"/><Raum = "SR1"/>
          <Zeit = "AB, Mittwoch, 7.30 - 9.00"/>
      <Aenderung Datum = "1.1.2000"><Vorschlag><Zeit><von>Montag</von>
          <in>Mittwoch</in></Vorschlag>
      </Aenderung></geplant>
  <Studiengang Name ="Informatik" Phase = "Hauptstudium"/>
  <Typus ID = "TypusID1"> <Art>Normalvorlesung</Art> <Umfang> 2+2+2 </Umfang> </Typus>
  <Kurs Name = "Datenbanken III"><Inhalt ID ="4711"> .... </Inhalt> ... </Kurs>
  <Semester>Sommersemester 2000, 10.4. 2000 - 15.7.2000</Semester>
  <Verantwortlicher><Lehrveranstaltung><Dozent Name = "beta"/> <Uebung Name = "Feyer"/>
                    <Praktikum Name ="Vestenicky"/></Lehrveranstaltung>
     <Planung><geplant durch>Fak.-Ref. Schenk</geplant durch>
             <Datum>1.4.1999, .... </Datum></Planung>
  <Vorschlag ID ="08015" Von = "KK"> <Sonderwunsch><Zeit>AB, Montag, 7.30-11.00</Zeit>
          <Raum><Ausstattung>Beamer, Netzanschlu&szlig;</Ausstattung><Raum>
          <Nicht-Parallel>Datenbanken I</Nicht-Parallel></Sonderwunsch></Vorschlag>
</Lehrveranstaltung>
```

Die erste Methode wird meist für die Speicherung und Verarbeitung in relationalen und objekt-relationalen DBMS angewandt. Die Repräsentation erfolgt auf der Grundlage von Sichten, die im Kapitel 6 ausführlich dargestellt werden. OLAP-Zugänge verwenden oft den zweiten Zugang. Die zweite Methode wird auch bei XML-DBMS angewandt.

Die Redundanz-Beherrschung ist nach wie vor für beliebige Objektmengen wichtig. Deshalb ist der erste Zugang vorzuziehen. Wir unterstützen diesen Zugang durch Einführung einer *Sichten-Suite*.

Motivations-schicht

Konzeptlandkarte

Konzept

Lastenheft: Daten

Vorstudie
Skizzierung

Geschäftsprozeß-schicht

Skizze

Grober Typ

Pflichtenheft: Daten

Feinstudie
Darstellung

Aktions-schicht

Skelett

Anwendungstyp

Anwendungsschema

Entwurf
Entwurf

konzeptionelle Schicht

Schema

Typ

ER-Schema

Implementation
Transformation

Implementations-schicht

Schema

logischer Typ

logisches Schema

Bild 25: Die Arbeitsprodukte im Abstraktionsschichtenmodell für die Strukturierung

5 Sprachen zur Darstellung der Funktionalität

> Ein Mann, der recht zu wirken denkt,
> Muß auf das beste Werkzeug halten.
> Bedenkt, Ihr habt weiches Holz zu spalten,
> Und seht nur hin, für wen Ihr schreibt.
> *Goethe, Faust, Vorspiel auf dem Theater, Direktor*

Die Herangehensweise zur Spezifikation der Funktionalität

Wir gehen von einer Einheit von statischen Gesichtspunkten (grundlegende Seiende und Beziehungen) und `dynamischen` Gesichtspunkten aus.
Dynamische Gesichtspunkte der Anwendung lassen sich spezifizieren durch

Operationen bzw. Handlungen zur Darstellung des dynamischen Verhaltens wie

- *Änderungsoperationen* zur Veränderung der Daten in der Datenbank,
- *Retrievaloperationen* zur Erschließung des Wissens aus der Datenbank ohne Veränderung der Datenbank,
- einer Sprache zur *Generierung* von Programmen und
- *Rollenveränderungen* von dargestellten Objekten.

dynamische Semantik auf der Grundlage von dynamischen Integritätsbedingungen zur Darstellung von *zugelassenen*, *erwarteten* und *verbotenen* Handlungsfolgen und

Verpflichtungen innerhalb einer *Rolle* beschreiben, welches Wissen zugänglich ist (Sichten) und welche Handlungsfolgen ausgeführt werden *müssen* (evt. unter Berücksichtigung von *Ursachen* (**aktive** Elemente)) bzw. *dürfen* (evt. unter Berücksichtigung von *Voraussetzungen*) sowie

Mitteilungen an einen oder den anderen *Empfänger*, die sie ihrerseits *verstehen* und *verarbeiten* können.

Veränderungen im Wissen müssen stets zu einer statischen Gesichtspunkten genügenden Aufzählung führen. Somit müssen Handlungen stets statisch abbildbar sein. Das Seiende ist etwas, das wirklich existiert. Es kann seine Existenz unabhängig von anderen beginnen und beenden. Damit werden formale Handlungen des Existenzbeginns und -endes grundlegend zur Umsetzung von Handlungen innerhalb der Datenbank.

Wir unterscheiden bei der Beschreibung der Funktionalität in der Modellierung zwischen

Produktfunktionen ***des Lastenheftes***, die in allgemeiner Form die Hauptfunktionen mit den Einschränkungen der Anwendung darstellen,

Arbeitsschritten ***des Pflichtenheftes***, die die Funktionalität auf dem Niveau der *Geschäftprozesse* und *Geschäftsvorfälle* in ihrem Ablauf unter Berücksichtigung der *Organisationsstruktur* darstellen,

Aktionen ***der Nutzer-Maschine*** zur vollständigen Beschreibung aller Handlungen von Benutzern aus deren Sicht,

Prozessen ***der Workflow-Maschine*** zur vollständigen konzeptionellen Spezifikation der Funktionalität der Anwendung
und

Programmen ***der Datenbank-Maschine*** auf dem Niveau der logischen Maschine bis hin zur Codierung von Stored Procedures, Triggern etc., die in Modulen zusammengefaßt werden.

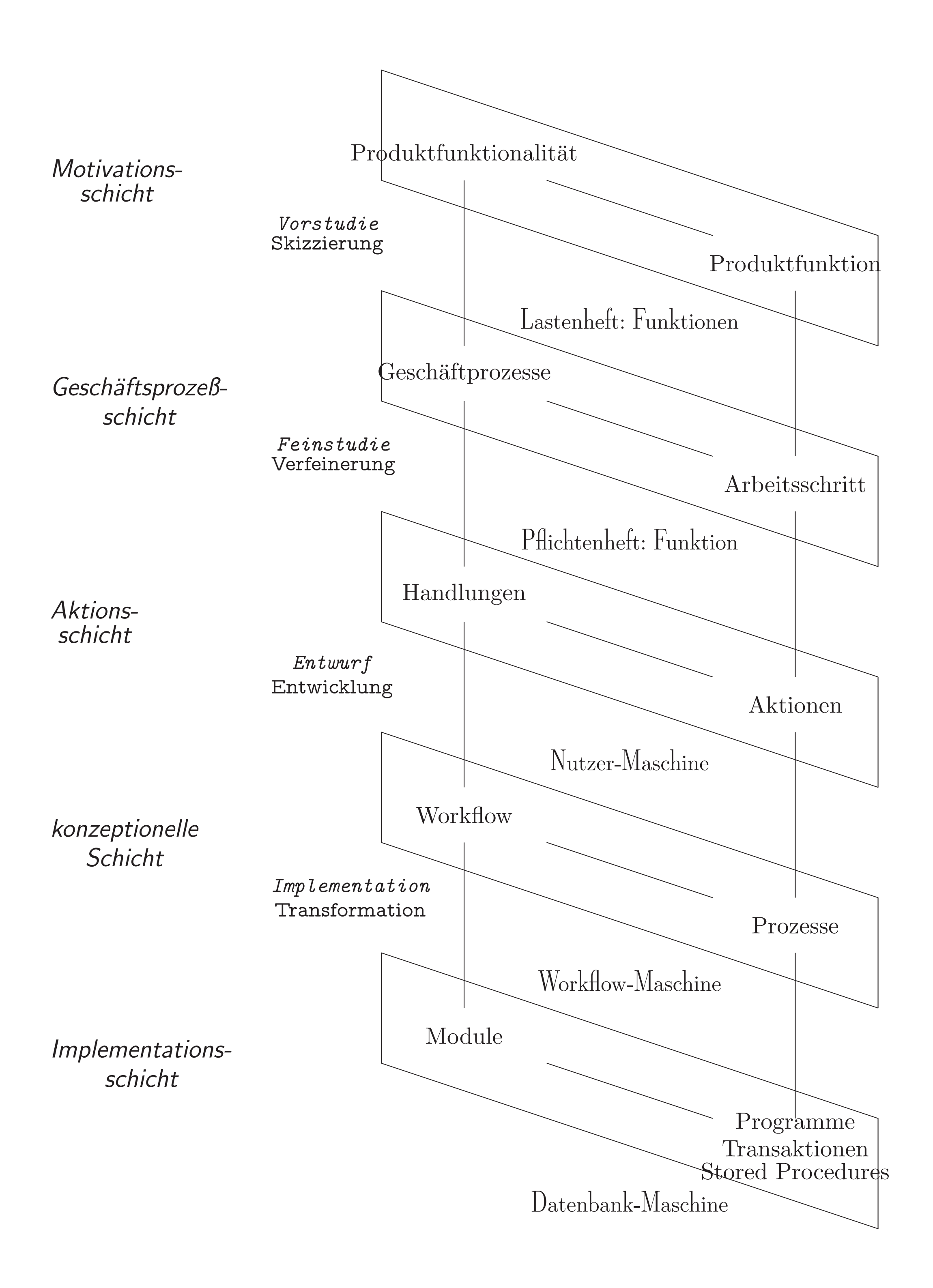


Bild 26: Die Arbeitsprodukte im Abstraktionsschichtenmodell für die Funktionalität

Die Abstraktionsschichten werden in Bild 26 illustriert.

Es existieren viele Modelle zur Darstellung der Prozesse und wenige Modelle zur Darstellung der dynamischen Semantik.

Formular-orientierte Sprachen erlauben die Modellierung von Folgen von zusammenhängenden Aktivitäten. Mit Ablaufmodellen kann der Lebenszyklus eines (Datenbank-)Objektes dargestellt werden. In einer *Form Definition Component* werden die Objekte selbst beschrieben. Mit der *Document Flow Component* wird der Datenfluß beschrieben. Die *Document Transformation Component* erlaubt die programmiersprachliche Beschreibung der Aktivitäten. Aktivitäten können selbst zu Gruppen zusammengefaßt werden. Verschiedene parallele Berechnungen sind möglich.

Fluß-orientierte Sprachen modellieren formale Handlungen als Flüsse von Objekten und Mitteilungen. Aktivitätengraphen bzw. Vorgangskettendiagramme, Prozeßmodelle und Beschreibungen von Lebenszyklen erlauben die Beschreibung von komplexem Verhalten.

Wir wählen hier einen ereignis-orientierten Zugang. Der Zusammenhang von Entities und Ereignissen wird durch Ereignisdiagramme in Petri-Netz-Darstellung (Ereignisse und Sichten) mit Input/Output in Eventknoten dargestellt. Knoten sind Ereignisse oder Sichten bzw. in einfachen Fällen Teilschemata. Kanten verlaufen von Ereignissen nach Sichten bzw. von Sichten nach Ereignissen. Sichten werden aufgefaßt als Input/Output-Generatoren. Ein Mehrfachinput kann in 'and'-Form für Ereignisse aufgefaßt werden. Ein Mehrfachinput an Sichten ist eine 'or'-Form zum Anstoßen. Damit ergibt sich eine Petri-Netz-Darstellung wie in Bild 27.

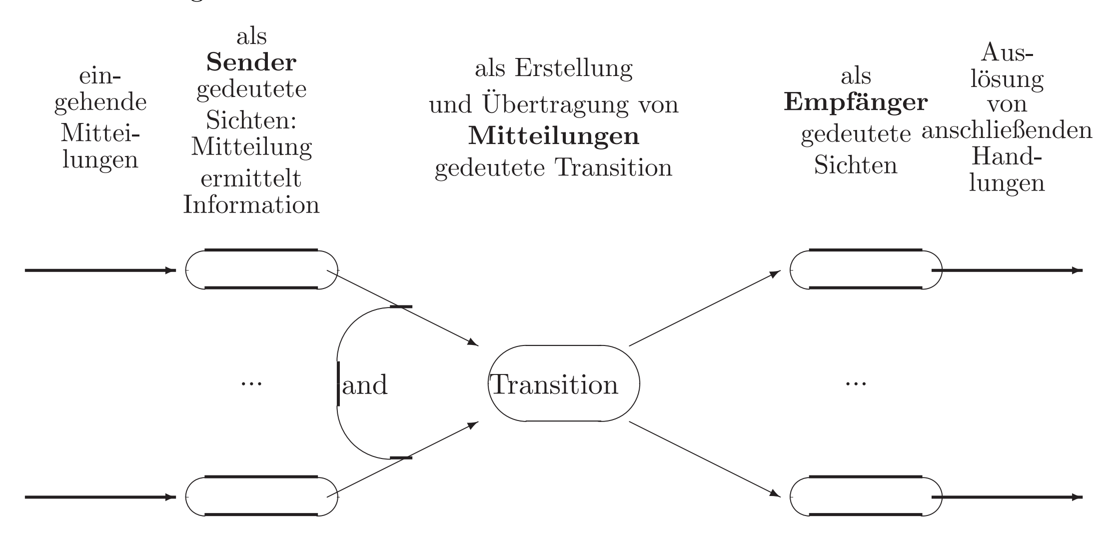


Bild 27: Petri-Netz-Darstellung von formalen Handlungen

Spezifikation der Geschäftsprozesse

Auf der Grundlage der Darstellung in Bild 27 können wir ein Aufgabenmodell entwickeln. Wir werden dieses Aufgabenmodell noch für die Spezifikation der Interaktivität durch Arbeitsvorgänge, Aktivitäten und Aktivitätenfolgendiagramme erweitern. Ein Arbeitsvorgang besitzt eine allgemeine Struktur, ein Resultat und semantische Rahmenbedingungen.

Ein Arbeitsvorgang im Rahmen eines Geschäftsprozesses wird durch

einen Namen,

einen Auslöser, der die Ausführung der im Arbeitsvorgang genannten Tätigkeiten auslöst (Zeitpunkte, eingehende Daten, Unterlagen, ...),

eine organisatorische Einheit, die eine Aufgabe durchführt,

eine Tätigkeit der Benutzer bzw. einen Ablauf von Tätigkeiten der Benutzer,

verwendete Hilfsmittel und Zusatzinformationen, die diese Tätigkeiten unterstützen, und

einer Ablage und einem Adressaten, in die oder an den Ausgaben erfolgen,

beschrieben.

Als Beispiel einer Aufgabe können wir die Erstellung eines Vorlesungsangebotes in unserem Hauptbeispiel betrachten. Ein Beauftragter eines Lehrstuhles erhält eine Aufforderung zur Erstellung von Angeboten zu einer Vorlesung. Die organisatorische Einheit ist der Lehrstuhl einer Universität. Hilfsinformation und Zusatzinformation sind die Angaben zu den angeforderten Kursen oder zu den neu angebotenen Kursen. Damit kann der Geschäftsvorfall so wie in Bild 28 dargestellt werden.

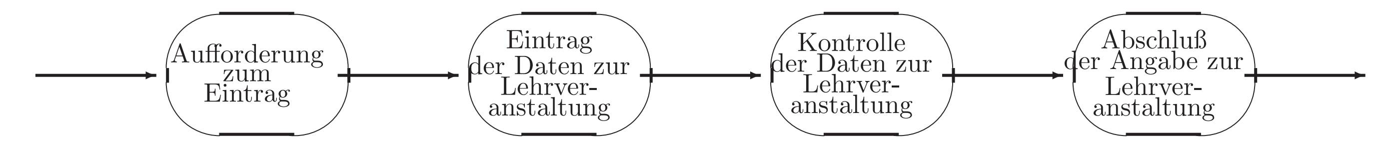


Bild 28: Geschäftsvorfall: Erstellung eines Angebotes für eine Lehrveranstaltung

Diese Graphik kann auch durch weitere Einzelschritte untersetzt werden. Anstelle der graphischen Darstellung kann auch eine tabellarische Darstellung gewählt werden:

Geschäftsvorfall: *Eintrag einer Lehrveranstaltung nach Aufforderung*		
Auslöser *Aufforderung für Beauftragten*	Organisatorische Einheit *Lehrstuhl*	Hilfs- und Zusatzinformation *Kurse, Studiengänge, Räume*
Tätigkeiten		für:
1. Eintrag		*Beauftragter des Lehrstuhles*
Hauptinformation angeben ; *(Klassifizieren || Einordnen || sonstige Angaben erfassen || Nebenbedingungen eingeben);*		
2. Kontrolle		*Beauftragter und Mitglieder des Lehrstuhles*
Informationsvergleich von Anforderungen, Angaben und weiteren Daten		
3. Beendigung		*Beauftragter des Lehrstuhles*
Ablegen am Lehrstuhl ; Absenden an auffordernde Einrichtung		

In analoger Form kann das Verhalten für Einzelobjekte durch eine Lebenszyklusspezifikation mit einem verallgemeinerten Petri-Netz-Modell (Prädikaten-Transitionsnetz) oder einem Automatengraphen beschrieben werden:

Menge von Zuständen: S_o für jedes Objekt in der Datenbank;

Menge von Aktivitäten: T_o für jedes Objekt in der Datenbank;

Diagramm: $D \quad \subseteq \quad S_o \times T_o \quad \cup \quad T_o \times S_o$;

Vor- und Nachbedingungen zu Aktivitäten: $V_o(s,t)$ für $(s,t) \in S_o \times T_o$ als Vorbedingung für eine Aktivität und $N_o(t,s)$ für $(t,s) \in S_o \times T_o$ als Nachbedingung für eine Aktivität. Damit kann eine Darstellung durch eine Hoare-Logik VtN verwendet werden.

Spezifikation eines Lebenszyklus: $(S_o, T_o, F_o, V_o, N_o)$

Nachteilig ist, daß dieser Zugang nur für Einzelobjekte geeignet ist. Durch komplexe Bedingungen kann auch Verknüpfung von Lebenszyklen beschrieben werden. Mitunter kann das Zusammenwirken von Objekten eine Komposition des Verhaltens verschiedener Objekte erfordern. Der Moment eines Lebenszyklus sind alle Eigenschaften des Objektes im Zustand s.

Die Spezifikation der Nutzer-Maschine

Die einzelnen Geschäftsprozesse werden nun mit ihrem Verlauf im Detail dargestellt. Sie können durch Ablaufdiagramme dargestellt werden. Handlungen sollen zu einer Veränderung der Datenbank führen oder dem Informationsgewinn der Akteure dienen. Akteure sind Abstraktionen von Benutzergruppen, wie z.B. der Beauftragten des Lehrstuhls. Wir entwickeln damit allgemeine *Änderungsoperationen*, *Retrievaloperationen* und *Operationen zur Veränderung von Rollen von Objekten* mit entsprechenden dynamischen Integritätsbedingungen. Es werden *zugelassene*, *erwünschte*, *verbotene* und *erzwungene* Handlungen dargestellt. Für die einzelnen Akteure gibt es *Verpflichtungen*.

Handlungen werden *Arbeitsvorgängen* bzw. *Tätigkeiten* zugeordnet. Ein Arbeitsvorgang ist - wie bereits dargestellt - durch einen Akteur oder eine Gruppe von Akteuren als Auslöser mit Rollen und Rechten, eine organisatorische Einheit, einer Beschreibung der *Aktionen* in ihrer Abfolge, eine Ordnung und ihren Beziehungen, die verwendeten Hilfsmittel und Informationen und die Ablage der Resultate charakterisiert.

Aus der Beschreibung der Koordination der Handlungen werden *dynamische Integritätsbedingungen* abgeleitet. Spezielle dynamische Integritätsbedingungen sind *Methodenregeln*, die Aussagen darüber festhalten, wie bestimmte Aktionen ausgeführt werden und welche Umgebung (Daten, Akteure, Dialoge) zu ihrer Ausführung notwendig ist. Durch *Zeitregeln*, *Ausführungshäufigkeiten* und *Ausführungspriorisierungen* werden die Zeitparameter festgelegt. *Entscheidungsregeln* spezifizieren im weiteren, welche Tätigkeit zu welchem Resultat führen muß, kann bzw. sollte. Wir können dazu Entscheidungstabellen benutzen. Es werden aus den *Geschäftsfalldaten*, d.h. den Daten, die während eines Geschäftsprozesses anfallen, und den *Geschäftsdialogen* entsprechende Entwurfsobligationen für andere Entwurfsschritte abgeleitet. Jeder Aktion können aktionsspezifische Integritätsbedingungen zugeordnet sein. Unter den Aktionen kann eine Ordnung existieren, die als kausale Abhängigkeit für parallelisierte Aktionen dargestellt wird.

Weiterhin werden den Handlungen verschiedene *Varianten* von Aktionen zugeordnet.

Wir verwenden dazu eine Erweiterung der tabellarischen Darstellung der Tabelle zu Geschäftsvorfällen von Seite 57. Eine graphische Darstellung wird in den Schritten zur Feinstudie aufgezeigt. Die tabellarische Darstellung stellt ein Kollaborationsdiagramm dar und beinhaltet die folgenden Angaben:

Handlungen des Akteurs		
Auslöser ...	Organisatorische Einheit ...	Hilfs- und Zusatzinformation ...
Integritätsbedingungen		
obligatorisch ...	erlaubt ...	verboten ...
Methodenregeln		
Ausführung ...	Umgebung ...	Zeitparameter ...
Aktionen des Akteurs		für:
i. Handlung		Akteur
Rechte ...	Pflichten ...	Rolle ...
i.j. Aktion ...		Integritätsbedingung

Im Detail stellt sich die Entfaltung der Tabelle von Seite 57 wie folgt dar:

<table>
<tr><td colspan="3">Handlungen des Akteurs: Eintrag einer Lehrveranstaltung nach Aufforderung</td></tr>
<tr><td>Auslöser
...</td><td>Organisatorische Einheit
...</td><td>Hilfs- und Zusatzinformation
...</td></tr>
<tr><td colspan="3">Integritätsbedingungen</td></tr>
<tr><td>obligatorisch
Beendigung_Bis_Termin</td><td>erlaubt
Entfernung von Angebot</td><td>verboten
Parallelangebot zu anderem Lehrstuhl</td></tr>
<tr><td colspan="3">Methodenregeln</td></tr>
<tr><td>Ausführung
Mit_Unterbrechung, Erinnerungsskripte, temporäre Ansicht, Gruppenarbeit, Erfolgsmeldung</td><td>Umgebung
Sitzungsobjekt, Online_Interface, konfigurierbare Oberfläche</td><td>Zeitparameter
Temporäre Ablage, wiederholter Aufruf, niedrige Priorität</td></tr>
<tr><td colspan="3">Aktionen des Akteurs für:</td></tr>
<tr><td colspan="3">1. Eintrag von Angeboten zu Lehrveranstaltungen — Beauftragter des Lehrstuhles</td></tr>
<tr><td>Rechte
Eintrag/Abschluß, Einsicht in Lehrstuhlangaben, Einsicht in Anforderungsliste, Eintragen, Streichen und Modifikation von Angeboten</td><td>Pflichten
vollständige Abarbeitung der Liste</td><td>Rolle
Datenbereitstellung</td></tr>
<tr><td colspan="3">1.1. Entgegennahme der Einzelaufgaben
1.1.1. Auswahl aus der Aufgabenliste — <code>DO UNTIL END_Of_LIST</code>
1.1.1.1. Lehrveranstaltungsidentifikation bestätigen
1.1.1.2. Auswahl der Lehrveranstaltungsart
1.1.1.3. Bestätigung oder Modifikation der Bezeichnung der Lehrveranstaltung
1.1.1.4. Bestätigung oder Modifikation der Inhaltsbeschreibung der Lehrveranstaltung
1.1.1.5. Zuordnung der Lehrenden zur Lehrveranstaltung
...
1.1.2. Angaben zur Art der Lehrveranstaltung — <code>parallel zu 1.1.3 ... 1.1.6</code>
1.1.2.1. Angaben zur Art der Durchführung
....
1.1.3. Bestätigung oder Modifikation der Zielgruppe für die Lehrveranstaltung
1.1.3.1. Bestätigung oder Modifikation des Studienganges
...
1.1.6. Angaben zur Nebenbedingungen der Lehrveranstaltung — <code>optional</code>
1.1.6.1. Angaben zu Terminwünschen
1.1.6.2. Angaben von Parallelveranstaltungen
....
1.3. Zusatzangaben zum Lehrbericht — <code>optional</code></td></tr>
</table>

Diese Tabelle kann als eine Auffaltung oder Verfeinerung der Tabelle von Seite 57 betrachtet werden. Damit sind wir in der Lage, die Konsistenz der Entwicklungsschritte direkt zu betrachten.

Die Algebra des erweiterten Entity-Relationship-Modelles

Alle Funktionen basieren auf einer entsprechenden Algebra, die zum einem Elementar-Operationen bereitstellt und zum anderen die Konstruktion von komplexeren Operationen auf der Grundlage vorhandener Operationen ermöglicht.

Wir erlauben eine komplexere Strukturierung von Typen. Deshalb verallgemeinern wir für die Definition von Operationen Teilstrukturen und Vergleiche:

Teilstrukturen von Strukturen und Typen sind mit folgenden Operationen definiert:

- Die Definition von Teilstrukturen basiert auf der Ordnung $\preceq$, die als kleinste binäre Relation über der Menge $\mathfrak{S}$ aller Strukturen definiert mit folgenden Eigenschaften:
  - $\lambda \preceq X$ für jedes $X \in \mathfrak{S}$;
  - $A(A_1, ..., A_n) \preceq B(B_1, ..., B_n)$ falls für alle i, $1 \leq i \leq n$ $\quad A_i \preceq B_i$ gilt;
  - $A\{A_1\} \preceq A\{B_1\}$ falls $A_1 \preceq B_1$ gilt.
- Die Vereinigung $Y \sqcup_X Z$, der Durchschnitt $Y \sqcap_X Z$ und die Differenz $Y \setminus_X Z$ sind dann für Strukturen X und deren Teilstrukturen Y, Z wie folgt induktiv definiert:
  - falls $Y \preceq Z$ gilt auch $Y \sqcup_X Z = Z$, sowie $Y \sqcap_X Z = Y$, $Z \setminus_X \lambda = Z$ und $Z \setminus_X Y = \lambda$;
  - falls $X = A\{B\},$, $Y = A\{C\}$, $Z = A\{D\}$ dann gilt auch $Y \circ_X Z = A\{C \circ_B D\}$ für $\circ \in \{\sqcup, \sqcap\}$;
  - falls $X = A\{B\}$, $Y = A\{C\}$, $Z = A\{D\}$, $Z \not\preceq Y$ dann gilt auch $Z \setminus_X Y = A\{D \circ_B X\}$;
  - falls $X = (A(A_1, ..., A_n)$, $Y = A(B_1, ..., B_n)$, $Z = (C_1, ..., C_n)$ dann gilt auch $A(B_1 \circ_{A_1} C_1, ..., B_n \circ_{A_n} C_n)$ für $\circ \in \{\sqcap, \sqcup, \setminus\}$.

  Die Struktur X wird als Kontext für die Operationen benötigt.

Prädikate: Gegeben sei ein Typ X. Ein Basisprädikat α vom Typ X ist ein Ausdruck der Form $Y \Theta a$ oder der Form $Y \circ C$ für $Y \preceq X$, $a \in dom(Y)$, $\circ \in \{\in, \notin\}$, $C \subseteq dom(Y)$ und alle Vergleichsprädikate Θ, die über Y definiert sind (Für viele Typen sind dies $\{=, \neq, <, >, \leq, \geq\}$.).
Ein Objekt o vom Typ X erfüllt $Y \Theta a$, falls $a \Theta o|_Y$ für die Einschränkung von o auf Y gilt.
Ein Objekt o vom Typ X erfüllt $Y \circ C$, falls $o|_Y \circ C$ für die Einschränkung von o auf Y gilt.
Prädikate sind induktiv definiert:

- Basisprädikate sind Prädikate.
- Sind α and β Prädikate, dann sind es auch $\alpha \wedge \beta$, $\alpha \vee \beta$ und $\neg\alpha$.

Ein Objekt o erfüllt das Prädikat α, falls dies entsprechend der Definition von α gilt. Damit definieren wir

$$\iota(\{o\}) = \begin{cases} \{o\} & \text{falls } o \text{ das Prädikat } \alpha \text{ erfüllt} \\ \emptyset & \text{andernfalls} \end{cases}$$

Erfüllt ein Objekt o das Prädikat α, dann

Ersetzungsfamilie: Eine Ersetzungsfamilie $\gamma = \{(o, R^{C_o})\}$ vom Typ R ist eine Menge bestehend aus einem Paar von Objekten und Klassen vom Typ R. Eine Ersetzungsfamilie beschreibt für Objekte vom Typ R jeweils eine Klasse von Objekten, durch die dieses Objekt ersetzt wird.

Definitionsrahmen der strukturellen Rekursion: Durch strukturelle Rekursion wird ein allgemeiner Rahmen zur Definition von Funktionen bereitgestellt.
Gegeben seien Typen T, T', Kollektionen T^C vom Typ T, die Funktionen $\cup_T$ (verallgemeinerte Vereinigung), $\cap_T$ (verallgemeinerter Durchschnitt) und die leere Kollektion $\emptyset_T$ von T.
Weiterhin seien für einen Typ T' ein Wert $h_0 \in dom(T')$ und Funktionen
$h_1 : T \to T'$ $\qquad h_2 : T' \times T' \to T'$ gegeben. Dann definieren wir

- $srec_{h_0,h_1,h_2}(\emptyset_T) = h_0$
- $srec_{h_0,h_1,h_2}(\{\!|s|\!\}) = h_1(s)$ $\qquad$ für einelementige Kollektionen $\{\!|s|\!\}$
- $srec_{h_0,h_1,h_2}(T^{C_1} \cup_T T^{C_2}) = h_2(srec_{h_0,h_1,h_2}(T^{C_1}), srec_{h_0,h_1,h_2}(T^{C_2}))$ falls $T^{C_1} \cap_T T^{C_2} = \emptyset_T$.

Gewöhnlich wird eine solche mathematische Definition weggelassen. Wir sind jedoch an einer Datenbankentwicklung mit nachvollziehbaren Eigenschaften interessiert.

Die Algebra des erweiterten ER-Modelles ist eine Verallgemeinerung und Erweiterung der relationalen Algebra. Demzufolge sind die Elementaroperationen auf die gleiche Art formulierbar:

- Typ-erhaltende Operationen führen zu Klassen vom gleichen Typ.

  Mengen-Operationen: Es seien R^{C_1} und R^{C_2} Klassen vom Typ R. Dann können wir folgende Operationen definieren:

  Vereinigung: $R^{C_1} \cup R^{C_2} \;=\; \{o \mid o \in R^{C_1} \vee o \in R^{C_2}\}$

  Durchschnitt: $R^{C_1} \cap R^{C_2} \;=\; \{o \mid o \in R^{C_1} \wedge o \in R^{C_2}\}$

  Differenz: $R^{C_1} \setminus R^{C_2} \;=\; \{o \mid o \in R^{C_1} \wedge o \notin R^{C_2}\}$

  Auswahl von Objekten aus einer Klasse: Gegeben sei ein Prädikat α über R.
  Die Selektion $\sigma_\alpha(R^C)$ wird induktiv eingeführt durch *strukturelle Rekursion* mit $\emptyset_T, \sqcup_R, \sqcap_R$ und
  $\sigma_\alpha(R^C) \;=\; srec_{\emptyset,\iota,\sqcup_R}(R^C)$ bzw. in der aufgelösten Form:

  - $\sigma_\alpha(\emptyset) \;=\; \emptyset$
  - $\sigma_\alpha(\{o\}) \;=\; \iota(\{o\})$
  - $\sigma_\alpha(R^{C_1} \sqcup_R R^{C_2}) \;=\; \sigma_\alpha(R^{C_1}) \sqcup_R \sigma_\alpha(R^{C_2})$ falls $R^{C_1} \sqcap_R R^{C_2} = \emptyset$ gilt.

  Wir nutzen eine andere Einführung als die viel verwendete doppelte Induktion wegen der komplexeren Teilstrukturierung der Typen. Die beiden Definitionen sind jedoch äquivalent.

  Abgeleitete Elementaroperationen sind die Modifikationsoperationen der Datenbanksysteme:

  Einfügen von Elementen: Die *insert*-Operation `Insert(`R^C, o`)` ist durch die Vereinigung $R^C \cup \{o\}$ von Mengen für Klassen R^C und Objekte o vom gleichen Typ R beschreibbar.

  Streichen von Elementen: Die *delete*-Operation `Delete(`R^C, o`)` ist durch die Differenz $R^C \setminus \{o\}$ von Mengen für Klassen R^C und Objekte o vom gleichen Typ R definierbar. Analog kann man auch das Streichen von Mengen `delete(`$R^C, R^{C'}$`)` einführen.

  Update von Elementen: Die Modifikation `Update(`R^C, α, γ`)` von Klassen R^C ist für Prädikate α und Ersetzungsfamilien $\gamma = \{(o, R^{C_o})\}$ ist definiert durch die Menge

  $$R^C \setminus \sigma_\alpha(R^C) \cup \bigcup_{o \in \sigma_\alpha(R^C)} R^{C_o} \quad .$$

  Eine oft verwendete Definition basiert auf dem Ausdruck $R^C \setminus \sigma_\alpha(R^C) \cup R^{C'}$. Damit wird jedoch ein anderer Effekt erzielt. Gilt z.B. $\sigma_\alpha(R^C) = \emptyset$ und $R^{C'} \neq \emptyset$, dann wird die ursprüngliche Intention verloren. Dieser Einführung liegt jedoch die oft praktizierte Ersetzung von `Update(`$R^C, o, \{o'\}$`)` durch die Folge `Delete(`R^C, o`);` `InsertUpdate(`R^C, o'`)` zugrunde.

- Typ-bildende Operationen erzeugen neue Klassen und Typen. Gegeben seien die Typen R und S und entsprechende Klassen R^C und S^C.

  Kartesisches Produkt: Die Klasse $R^C \times S^C \;=\; \{(o, o') \mid o \in R^C, o' \in S^C\}$ ist definiert über dem Typ (R, S).
  Das Kartesische Produkt kann auch in entfalteter Form über eine Konkatenation der Objekte gebildet werden.

  Projektion: Es sei R_1 ein Teiltyp von R. Die Projektion $\Pi_{R_1}(R^C)$ von R^C auf R_1 durch eine Reduktion aller Objekte von R^C auf den Typ R_1.
  Mitunter wird auch die Notation $R^C[R_1]$ anstelle von $\Pi_{R_1}(R^C)$ verwendet.

  Schachtelung: Es sei R' ein Element von R. Dann wird die Schachtelung $\nu_{R'}(R^C)$ von R^C entlang von R' definiert als Klasse über dem Typ $T = (R \setminus R') \sqcup_R \{R'\}$ mit der Menge von Objekten
  $\{\, o \in Dom(T) \mid \exists o' \in R^C : o[R \setminus_R R'] = o'[R \setminus_R R']$
  $\wedge\, o(R') \;=\; \{\, o''[R'] \mid o'' \in R^C \wedge o'[R \setminus_R X] = t''[R \setminus_R R']\}\}.$

  Entschachtelung: Es sei R' ein Mengenelement von R. Die Entschachtelung $\mu'_R(R^C)$ einer Klasse definiert einen neuen Typen $T = (R \setminus_R \{R'\}) \circ R'$ für die Konkatenation $\circ$ und die neue Klasse
  $\{\, o \in Dom(T) \mid \exists o' \in R^C : o[R \setminus_R \{R'\}] = o'[R \setminus_R \{R'\}] \wedge o[X] \in o'(X)\}$

Potenzmenge: Die Potenzmenge $\mathcal{P}(R^C) = \{M | M \subseteq R^C\}$ ist eine geschachtelte Klasse über dem Typ $\{R\}$.

Umbenennung: Gegeben sei ein Typ R. Es sei ϕ eine bijektive Abbildung auf der Markierungsmenge L mit der Einschränkung, daß für Namen A, B in R mit $\phi(A) = B$ auch $dom(A) = dom(B)$ gelten muß. Die Umbenennung von R mit ϕ bildet die Klasse R^C auf eine Klasse $\rho_\phi(R^C) = \phi(R^C)$ über $\phi(R)$ ab.

Verbund: Der Verbund $R^{C_1} \bowtie S^{C_2}$ ist definiert durch den Ausdruck $\Pi_{R \sqcup S}(\sigma_{R \sqcap S \in \Pi_{R \sqcap S}(S^{C_2})}(R^{C_1}) \times S^{C_2}$.

Aggregationsoperationen werden in OLAP-Anwendungen vielfältig angewandt. Eine Aggregationsoperation ist definiert als Familie $\mathcal{F} = \{f_0, ..., f_k, ..., f_\omega\}$ mit Funktionen $f_k : Bag_k \rightarrow Num$, die Multimengen mit k Elementen vom Typ T auf einen numerischen Datentyp Num abbilden. Wir lassen nur solche Typen zu, die ein minimales und ein maximales Element in $dom(T)$ besitzen. Es müssen zwei Eigenschaften bezüglich der Ordnung auf $dom(T)$ erfüllt sein:

- Es gelten die Gleichungen $f_k(min, ..., min) = min$ und $f_k(max, ..., max) = max$ für die minimalen und maximalen Elemente in $dom(T)$.
- Die Funktionen sind monoton bzgl. der Ordnung von $dom(T)$.

Da Nullwerte explizit zugelassen sind, benutzen wir zwei Hilfsfunktionen für die strukturelle Rekursion:

$$h_f^0(s) = \begin{cases} 0 & \text{falls } s = \text{NULL} \\ f(s) & \text{falls } s \neq \text{NULL} \end{cases}$$

$$h_f^{\text{undef}}(s) = \begin{cases} \text{undef} & \text{falls } s = \text{NULL} \\ f(s) & \text{falls } s \neq \text{NULL} \end{cases} .$$

Wir können nun die folgenden üblichen Aggregationsfunktionen einführen:

Summierung in unterschiedlichen Varianten abhängig von der Behandlung von Nullwerten:

- Summierung für Klassen ohne Nullwerte:
  $\texttt{sum} = srec_{0,Id,+}$;
- Summierung für Klassen mit Nullwerten, die durch die 0 ersetzt werden:
  $\texttt{sum}_0^{null} = srec_{0,h^0_{Id},+}$;
- Summierung für Klassen mit Nullwerten, die durch die $undef$ ersetzt werden:
  $\texttt{sum}_{undef}^{null} = srec_{0,h^{\text{undef}}_{Id},+}$.

Üblich ist die Anwendung der zweiten Option.

Zählen der Objekte je nach Behandlung der Nullwerte:

- Für Klassen ohne Nullwerte: $\texttt{count} = srec_{0,1,+}$;
- Für Klassen mit Nullwerten: $\texttt{count}_1^{null} = srec_{0,h^0_1,+}$;
- Alternativ für Klassen mit Nullwerten: $\texttt{count}_{undef}^{null} = srec_{0,h^{\text{undef}}_1,+}$.

Genutzt wird oft die zweite Option.

Bildung der maximalen bzw. minimalen Werte in Abhängigkeit von den Ordnungen für NULL:

- Die leere Menge erlaubt keine Bestimmung von minimalen bzw. maximalen Werten:
  - $\texttt{max}_{\text{NULL}} = srec_{\text{NULL},Id,max}$ bzw. $\texttt{min}_{\text{NULL}} = srec_{\text{NULL},Id,min}$
  - $\texttt{max}_{\text{undef}} = srec_{\text{undef},Id,max}$ bzw. $\texttt{min}_{\text{undef}} = srec_{\text{undef},Id,min}$

  Diese Funktionen hängen davon ab, wie die Nullwerte in $dom(T)$ eingeordnet werden.

Bildung des Durchschnittes: Die Durchschnittsbildung ist eine komplexere Funktion. Es gibt dafür eine Reihe von Möglichkeiten:

$$(++)\ \frac{\texttt{sum}}{\texttt{count}} \qquad (??)\ \frac{\texttt{sum}}{\texttt{count}^{null}} \qquad (??)\ \frac{\texttt{sum}}{\texttt{count}^{null}}$$

$$\text{(SQL!?)}\ \frac{\mathtt{sum}_{0}^{null}}{\mathtt{count}} \qquad (+!)\ \frac{\mathtt{sum}_{0}^{null}}{\mathtt{count}_{1}^{null}} \qquad (??)\ \frac{\mathtt{sum}_{0}^{null}}{\mathtt{count}_{undef}^{null}}$$

$$(+?!)\ \frac{\mathtt{sum}_{undef}^{null}}{\mathtt{count}} \qquad (??)\ \frac{\mathtt{sum}_{undef}^{null}}{\mathtt{count}_{1}^{null}} \qquad (++)\ \frac{\mathtt{sum}_{undef}^{null}}{\mathtt{count}_{undef}^{null}}$$

SQL benutzt eine Variante, die nicht die intuitivste ist. Wir präferieren in der HERM-Algebra die mit "+" annotierten Varianten für den Fall von Klassen mit Nullwerten. Die Funktionen $\mathtt{avg}_{0,1}^{null}$ und $\mathtt{avg}_{undef}^{null}$ werden dabei der SQL-Form $\mathtt{avg}^{null}$ vorgezogen.

Die algebraischen Operationen können zur Bildung von komplexeren Ausdrücken benutzt werden. Eine HERM-Anfrage ist ein (komplexer) Ausdruck in der HERM-Algebra.

Erweiterte HERM-Spezifikation von Operationen

Wir erweitern die HERM-Algebra um die Spezifikation einer Umgebung (Sicht), Ausführungsbedingungen als Vorbedingung und Nachbedingung und Nachfolgeoperationen. In diesem Fall erhalten wir einen allgemeinen Definitionsrahmen der Form:

Operation φ
[`Sicht:` < Sichten_Name>]
[`Vorbedingung:` < Aktivierungs_Bedingung >]
[`Aktivierte_Operation:` < Spezifikation >]
[`Nachbedingung:` < Akzeptanz_Bedingung >]
[`Erzwingene_Operation:` < Operation, Bedingung>]

Sprachen zur Darstellung von Workflows

Die Arbeitsvorgänge werden in Einzelschritte zerlegt. Die Einzelschritte werden durch *Prozesse* verfeinert. Der Zusammenhang der Arbeitsvorgänge wird durch verallgemeinerte Transaktionsmodelle dargestellt. In der Datenbank verändern Prozesse die *Zustände*. Deshalb werden die Zustandsveränderungen modelliert. Die Prozesse veranlassen legale Transitionen von Zuständen. Darauf aufbauend können die Integritätsbedingungen durch Bedingungen zur Ausführung und durch Nachbedingungen für Prozesse dargestellt werden. Integritätsbedingungen, die durch Transitionsbedingungen nicht darstellbar sind, werden für Pflegeroutinen aufbereitet. Mit der Prozeßdefinition kann die Definition der Semantik abgeleitet werden. Je nach Prozeßmodell wird eine axiomatische, parallele, kausale etc. Semantik benutzt. Wir benutzen ein Zustandstransitionsdiagramm zur Darstellung.

Die Aufgaben- und Prozeßkoordination folgt den Zusammenhängen in den Geschäftprozessen.

Wir unterscheiden für die Prozesse *Retrievaldaten*, die als Input für die Prozesse aus der Datenbank dienen, *Inputdaten* der Akteure, *Outputdaten* zum Zurückschreiben in die Datenbank, *Displaydaten* zur Darstellung in den Dialogen und *Begleitdaten*, die aus vorhergehenden Prozessen stammen und zur Darstellung der Informationen während der Dialogschritte dienen. Damit können Prozesse auch einander beeinflussen und sind nicht nebenwirkungsfrei. Damit werden für die Prozesse auch *Interaktionsdiagramme* und *Kohäsionsbeziehungen* entwickelt.

Damit erhalten wir ein allgemeines Workflow-Modell:

Die drei R's von Workflow-Modellen sind

Routen (Szenario) durch einen Ablaufgraph,

Regeln zur Darstellung der verarbeiteten Information und

Rollen mit einer Zuordnung zu den handelnden Personen (Akteuren).

Die drei P's von Workflow-Modellen sind

Prozesse als das Kernstück der Spezifikation,

Politiken und Anwendungsstrategien und

Praktiken der Anwendung, die spezifische Seiten zum Ausdruck bringen.

Für die Verhaltensmodellierung ergeben sich neue Modellierungsforderungen:

- Erweiterte und abgeschwächte Transaktionsmodelle können verwendet werden. Dazu stehen als Alternativen Konzepte des Transaktionsbaumes, genesteter Transaktionen, offene genestete Transaktionen und kompensierende Teiltransaktionen zur Verfügung.

  Das erweiterte ER-Modell verfügt über diese Mechanismen.
  Es wird eine Transaktion allgemein mit einem Definitionsrahmen der Form
  `transaction` *identifier (parameter list)*
  $o_1; o_2; ...; o_m$
  `end`;
  angegeben. Die Operationen o_i sind HERM-Operationen. Sie können parametrisiert sein.

  Weiterhin sind geschachtelte Transaktionen $P_1; P_2; ...P_n$ zugelassen, die ihrerseits wiederum aus Folgen von (Komponenten-)Transaktionen bestehen. Die Semantik der geschachtelten Transaktionen basiert auf einem schrittweisen Abschluß der Komponenten-Transaktionen. Führt eine der Komponenten-Transaktionen zu einem Fehler, dann wird die gesamte geschachtelte Transaktionen abgebrochen.

  Außerdem sind zugelassen

  - *kompensierende Transaktionen*
    P_1 `compensated_by` P_2,
    bei denen bei einem Auftreten eines Fehlers die Transaktion zu einer Kompensation des Fehlers benutzt wird und die Transaktion erst dann abgebrochen wird, wenn auch die kompensierende Transaktion nicht zum Erfolg führt, sowie
  - *Ersatztransaktionen*
    P_1 `contingented_by` P_2,
    bei denen bei Auftreten eines Fehlers der Transaktion P_1 die Transaktion P_1 auf den Beginn zurückgeführt wird und anschließend P_2 ausgeführt wird, so daß die Gesamttransaktion nur dann abgebrochen wird, wenn sowohl P_1 als auch P_2 nicht zum Erfolg führen.

  Eine (einfache) Transaktion ist eine Folge $P = o_1; o_2; o_3; ...; o_m$ von Basismodifikations- und Retrieval-Operationen über dem Datenbankschema $\mathcal{ER} = (\{T_i | 1 \leq i \leq m\}, \Sigma_{\mathcal{ER}})$ mit $T_i = (U_{T_I}, \Sigma_{T_i})$ für $1 \leq i \leq m$.
  Transaktionen können auf einen Datenbank-Zustand $\mathcal{ER}^C$ sequentiell angewandt werden und führen zu einer Transition $o_m(...(o_2(o_1(\mathcal{ER}^C))))$.

  Der Effekt der Anwendung einer Transaktion P auf $\mathcal{ER}^C$ ist definiert als *Transformation, die die Integritätsbedingungen erhält*, d.h.

$$P(\mathcal{ER}^C) = \begin{cases} o_m(...(o_2(o_1(\mathcal{ER}^C)))) & \text{falls} \quad o_m(...(o_2(o_1(\mathcal{ER}^C)))) \models \Sigma_{\mathcal{ER}} \cup \bigcup_{i=1}^{m} \Sigma_{T_i} \\ \mathcal{ER}^C & \text{andernfalls} \end{cases}$$

  Damit kann eine Transaktion als integritätsinvariante oder konsistenzerhaltende Zustandstransformation verstanden werden.

  Sie stellen daher eine besondere Form von Programmen dar.

  Wir nutzen als Ausführungsmodell das Zustandsmodell in Bild 29. Eine Transaktion ist entweder inaktiv oder aktiviert oder beendet (EOT). Eine aktivierte Transaktion ist beim Bereitstellen aller benötigten Ressourcen (BOT) oder in der Bearbeitung oder bereit zum Abschluß (Commit), wobei dann die Gültigkeit der statischen und dynamischen Integritätsbedingungen geprüft werden muß oder beim Zurückfahren zum inaktiven Zustand, falls die Prüfung der Konsistenz eine Inkonsistenz ergeben hat, oder beim Abschluß, wobei dann alle Ressourcen wieder freigegeben werden.

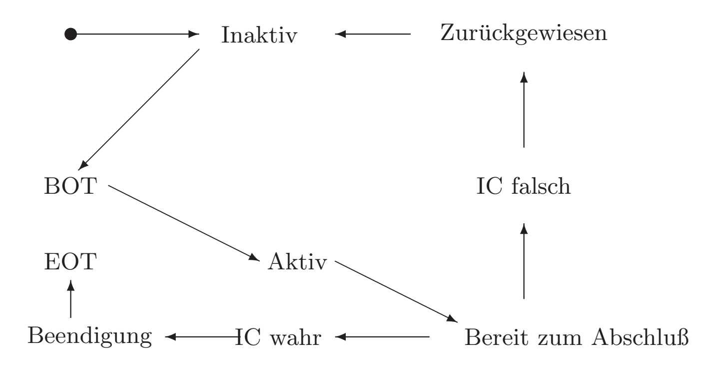


Bild 29: Die Zustände einer Transaktion

Zur Implementation von Transaktionssystemen steht eine Reihe von Update-Optionen, wie Update-in-place (auf der Platte), Update-in-private (mit einem Transaktionspuffer) oder Update-im-Schattenspeicher zur Verfügung.

Das klassische Transaktionsmodell definiert Transaktionen über das ACID-Konzept. Transaktionen sind atomar (werden ganz oder gar nicht wirksam), konsistenzerhaltend, werden isoliert ausgeführt und führen zu dauerhaften Veränderungen. Diese Auffassung postuliert die Existenz einer implementierenden Maschine. Auf der konzeptionellen Schicht können wir sie deshalb nicht verwenden. Die beiden ersten Bedingungen sind in unsere Definition eingeflossen. Die letzte Bedingung ist eine Forderung an Informationssysteme im allgemeinen. Die Isoliertheit wird gewöhnlich über die Serialisierbarkeit definiert. Da dies jedoch auch ein Implementationskonzept ist, verwenden wir einen anderen Zugang.

Die Read-Write-Mengen $read - write(P) = (read(P), write(P))$ einer Transaktion sind alle elementaren Lese- und Schreiboperationen der Transaktion P. Eine read-Operation ist eine objekt-basierte Operation, ebenso wie eine write-Operation.
Zwei Transaktion P_1 und P_2 sind *Konkurrenten* falls $read(P_1) \cap write(P_2) \neq \emptyset$ oder $read(P_2) \cap write(P_1) \neq \emptyset$ oder $write(P_2) \cap write(P_1) \neq \emptyset$.

Parallele Ausführung von Transaktionen ist immer möglich, wenn diese keine Konkurrenten sind. In diesem Fall ist der Effekt der parallelen Ausführung äquivalent zu $P_1(P_2(\mathcal{ER}^C))$ oder zu $P_2(P_1(\mathcal{ER}^C))$ für die Datenbank $\mathcal{ER}^C$.
Sind Transaktionen Konkurrenten, dann kann ein Steuerprogramm die Korrektheit der parallelen Ausführung gewährleisten.

Neben diesen Modellen können wir auch erweiterte Konzepte aus der Literatur verwenden:

- *Sagas* basieren auf einer Menge von ACID Subtransaktionen mit vordefinierter Ausführungsordnung und einer Menge dazu assoziierter kompensierender Teiltransaktionen.
- *Split-and-join-Transaktionen* wurden für den Ressourcentransfer zu parallel verlaufenden Transaktionen entwickelt.
- *Flexible Transaktionen* sind *Polytransaktionen*, deren Konsistenzforderungen durch Kollektion von Datenabhängigkeitsdeskriptoren (D^3) realisiert werden.
- Transaktionen können mit dem ACTA Metamodell erweitert werden.

Für unsere Belange erscheinen jedoch diese HERM-TA-Formen ausreichend.

- Es wurden unterschiedliche Modelle zur Ausführung und Steuerung von Geschäftsprozessen, Handlungen und Workflows entwickelt. Das einfachste Modell ist das Modell der *Job Control Language* (JCL). Dieses Modell wurde für *Skriptsprachen* erweitert. Eine Transaktion kann ebenso wie ein Modul als *abstraktes Programm* mit einem *Namen* und *formalen Parametern* für den *Input*, *Output* und *Reserven* zum Programmdurchlauf aufgefaßt werden.

Jedem abstrakten Programm sind *Parameter-Werte-Paare* zugeordnet, die entweder zur Aufrufspezifikation oder zur Steuerungsspezifikation herangezogen werden. Diese Parameter sind entweder ereignisbasiert oder wertebasiert. Wir verwenden solche Ereignisse oder Werteparameter in der konzeptionellen Schicht, um einen allgemeinen Rahmen für die Implementationsschicht schaffen zu können. Zu solchen Parametern gehören

- *allgemeine Aufrufparameter* `onLoad`, `onSelect`, `onSubmit`, `onUnload`, `onWait`, `onUnWait`, `getData`, `instantiateSession`, `presentationMode`, `presentationStyle`, `typeGlobeSelect`, `onBlur`, `onCancel`,
- *allgemeine Priorisierungsparameter* wie `onFocus`, `changePriority`, `unFocus`, `emphasisMode`,
- *allgemeine Steuerparameter* wie `onReset`, `onRecovery`, `onChange`, `onUserReaction`, `changeToSlave`, `changeToMaster`, `waitUntil`, `securityLevel`, `changeStatus`, `openSatelliteWindow`, `closeSatelliteWindow`, `hookOnProcess`, `separateFromProcess`, `defaultSet`, `onScroll`, `deliveryRestriction`, `deliveryMode`, `securityLevel`, `garbageCollector`, `hideMode`,
- *Fehlerparameter* wie `onAbort`, `onError`, `unloadErrorLog`, `useErrorLog` `notifyMode` und
- *allgemeine Weitergabeparameter* wie `onSend`, `onReceive`, `forUser`, `toNode`, `fromNode`, `onTime`, `validUntil`, `onLoad`, `onEventTransfer`, `onMouseCapture`, `whoCausedEvent` und `returnValue`.

Die Parameterliste kann beliebig verkleinert oder vergrößert werden. Im letzteren Fall müssen adäquate Formen für die Umsetzung in die Implementation gefunden werden.

Außerdem unterscheiden wir verschiedene Variablentypen:

- *Statische Variablen* sind analog zu den globalen Variablen von Pascal oder den Klassenvariablen von Java mit einer an das Programm gekoppelten Lebenszeit ausgestattet. Statische Variablen können *global* oder *lokal*. Per Default sind sie lokal.
- *Kellervariable* besitzen ebenso wie lokale Variablen oder Parametervariablen nur eine Lebenszeit während der Ausführung einer Funktion.
- *Explizite Speichervariablen* nutzen einen temporär zugeordneten Speicher.
- *Implizite Speichervariablen* werden erst mit einem Speicherplatz verbunden, wenn ihnen ein Wert zugewiesen wird.

In analoger Form können der *Gültigkeitsbereich* und die *Bindung* an Namen, Stellen und Werte (zur Entwurfszeit, Implementationszeit, Übersetzungszeit, Linkzeit, Ladezeit oder Laufzeit) von Variablen und Parametern behandelt werden.

Wir verwenden diese Konzepte zur Spezifikation des Aufrufes einer Transaktion und der Steuerung der Ausführung.

Aufrufsspezifikation: Ein Aufruf eines abstrakten Programmes wird über den Namen des Programmes, durch eine Instantiierung der formalen Parameter durch *aktuelle Parameter* und durch die Angabe des *Nutzers* und der *Steuerungsumgebung* angegeben. Die Angabe des Nutzers, der ein anderes Programm oder ein Benutzer sein kann, erfolgt nicht nur für Abrechnungs- sondern auch für Benachrichtigungszwecke. Die Steuerungsumgebung erlaubt eine detaillierte Angabe der Steuerung, der Priorisierung, der Ausführung auf anderen Knoten. Zur Angabe kann außerdem eine Spezifikation der *Ausnahmenbehandlung* gehören wie z.B. für den Fall eine konkurrierenden Abarbeitung.

Steuerungsspezifikation: Zur Steuerungsspezifikation unterscheiden wir drei Räume:

- Im *Nachrichtenraum* werden sowohl die Benutzernachrichten als auch die Systemnachrichten verwaltet. Im ersteren Falle sind Informationen zur Sichtbarkeit, Notifikation und zum Datenstrom des Benutzers, im letzteren zur Übertragung, Signoff- und Signon-Nachrichten gesetzt.

- Im *Parameterraum* werden alle aktuellen Parameterzuweisungen wie Status, Priorisierung, Ausführungsmodi (stand alone, eingebettet, remote, applet, servlet), Bindungen und Einschränkungen verwaltet. Außerdem erfolgt hier die Speicherung des Benutzerportfolios und des aktuellen Benutzerprofils.
- Im *Ressourcenraum* wird die Allokation von Daten, Routen, Knoten etc. zu den aktuellen Prozessen dargestellt.

Weitere Modelle sind möglich:

- *ConTracts* sind komplexere Modelle und geeignet für die Gruppierung von Transaktionen in eine Multitransaktionsaktivität (Menge von ACID Aktionen (Schritte) mit explizit spezifiziertem Ausführungsplan (Skript)), wobei die Ausführung Vorwärts-Recoverable sein muß (abgeschwächte atomicity)).
- Langlebige Aktivitäten (*Long running activities*) basieren auf einem erweiterten ECA Modell. Sie verwenden eine Menge von Ausführungsschritten, die (rekursiv) andere Transaktionen enthalten, und Kontroll- und Datenfluß-Skripte. Es wird eine explizite Kompensation für Transaktionen vorgegeben. Das Konzept wird durch eine Kommunikation zwischen den Ausführungsschritten unterstützt. Es werden außerdem die Abfrage des Status einer Aktivität und explizite Ausnahmebehandlung unterstützt. Es können Korrektheits- und Koordinierungsbedingungen angegeben werden. Daraus werden Aufgabenflußmodelle für Multiaktivitäten abgeleitet.

Damit umfaßt die Spezifikation eines Workflows

die Aufgaben- bzw. Prozeßspezifikation als spezifische Art eines Prozesses bei Spezifikation der Struktur, wobei

- die Menge von extern sichtbaren Ausführungszuständen,
- die Menge von legalen Transitionen dieser Zustände und
- Bedingungen, die die Ausführung der Transitionen erlauben,

für die Darstellung durch Zustandstransitionsdiagramme benutzt werden. Jeder Prozeß hat eine interne Struktur und ist damit abhängig vom Input und dem Zustand des lokalen Systemes. Er ändert den Zustand und produziert einen Output abhängig von verschiedenen Systemcharakteristika, abhängig von Eigenschaften der Operationen (z.B. analoge Ausführung, serielle Ordnung, Idempotenz von Operationen (günstig für Kompensation (z.B. setze x to c))),

die Aufgaben- bzw. Prozeßkoordination durch verschiedene Scheduling-(Pre-)Conditions

statisch durch Definition des Zustandes vor Start des Prozesses (Ausführungszustände anderer Prozesse, Output-Werte anderer Prozesse, Werte externer Variable) oder

dynamisch durch Spezifikation der Abhängigkeiten während der Prozeßausführung,

die Korrektheitsbedingungen für Ausführung und Versagen mit

Failure-atomicity Bedingungen (Bei Ausführungsproblemen kann anstatt des vorgegebenen Prozesses ein anderer ausgeführt werden, um einen akzeptablen Endzustand zu erreichen.) oder

Execution-atomicity Bedingungen (zur Darstellung der Zerlegbarkeit von Transaktionen bei Serialisierung).

Die Ausführung eines Workflows hängt von verschiedenen Interprozeß-Abhängigkeiten ab. Damit spezifizieren wir zwei Bestandteile:

den Scheduler zur Ausführung der Prozesse durch Planung der nächsten Schritte mit einem Monitoring der verschiedenen Ereignisse und zur Berechnung der Interprozeß-Abhängigkeiten und

den Prozeß-Agenten zur Ausführungskontrolle eines Prozesses.

Programme der Workflow-Maschine

Elementarprogramme sind alle zugelassenen Ausdrücke der HERM-Algebra. Wir unterlegen dabei eine Semantik der Abstract-State-Machines. Sie wird im folgenden kurz eingeführt. In diesem Buch werden wir die Semantik nur anwenden, so daß wir auf eine detaillierte Erklärung verzichten können. Für die graphische Darstellung schließen wir uns den üblichen Darstellungsformen für sequentielle Programme an, wobei wir eine Verwechslung mit Konstrukten des ER-Modelles vermeiden wollen. Deshalb sind ovale Boxen sowohl Programmschritten als auch dem Test vorbehalten. Wir können damit induktiv komplexere Programme konstruieren:

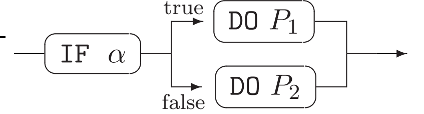
Sequentielle Ausführung von Programmen — DO $P_1; P_2$

Ein Programmschritt kann auf einen anderen Programmschritt folgen.

DO P_1 → DO P_2 →

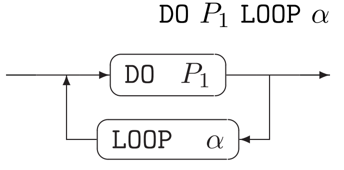
Verzweigung mit einer logischen Bedingung: — **if** α **then** P_1 **else** P_2;

Ein Programmschritt kann verzweigen unter einer Bedingung.



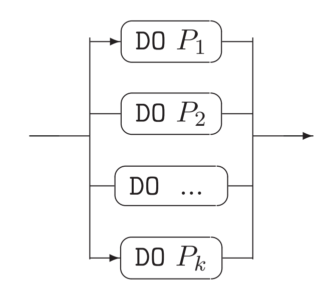
Wiederholte Ausführung mit einer logischen Bedingung: — DO P_1 LOOP α

Der Bequemlichkeit halber führen wir auch eine Programmschleife ein. Diese ist auch durch andere Konstrukte abstrakter Maschinen ausdrückbar.



Parallele Ausführung mehrerer Programme ohne Einschränkung: — DO P_1 PAR ... PAR DO P_k

Programme können parallel ausgeführt werden. Die parallele Ausführung ist beendet, wenn alle Programme beendet sind. Alle Programme beginnen mit dem gleichen Zustandsraum. Eine parallele Ausführung ist erfolgreich durchgeführt, wenn keine konkurrierenden Veränderungen der Datenbank mit unterschiedlichen Werten für die gleichen Datenbankobjekte erfolgen.



Ausführung nach Zuweisung von Werten zu Variablen: — LET $x = t$ IN P DO P

Es können Werte den Parametern in einem Programm P zugewiesen werden. Diese Zuweisung gilt nur lokal.

— LET $x = t$ IN P → DO P →

Ausführung nach Auswahl eines Wertes : — CHOOSE x WITH ϕ DO P

Es wird ein x-Wert unter einer Bedingung gewählt. Damit wird das Programm ausgeführt.

— CHOOSE x WITH ϕ → DO P →

Ausführung für alle zutreffenden Werte: — FOR ALL x WITH ϕ DO P

Alle Werte für einen Parameter, die zutreffen, werden gewählt. Es wird dafür das Programm P parallel ausgeführt.

— FOR ALL x WITH ϕ → DO P →

SKIP-Programmschritt zur Darstellung des leeren Programmschrittes: — SKIP

Konzeptionell kann auch der Programmschritt ohne Auswirkungen auf den Zustand benötigt werden.

— DO SKIP →

Modifikationsschritt zur Durchführung einer Modifikation der Datenbank: DO
$f(s_1, ..., s_n) := t$

Es wird ein paralleler Update für eine Datenbank-Klasse ausgeführt mit den Parametern $s_1, ..., s_n$.

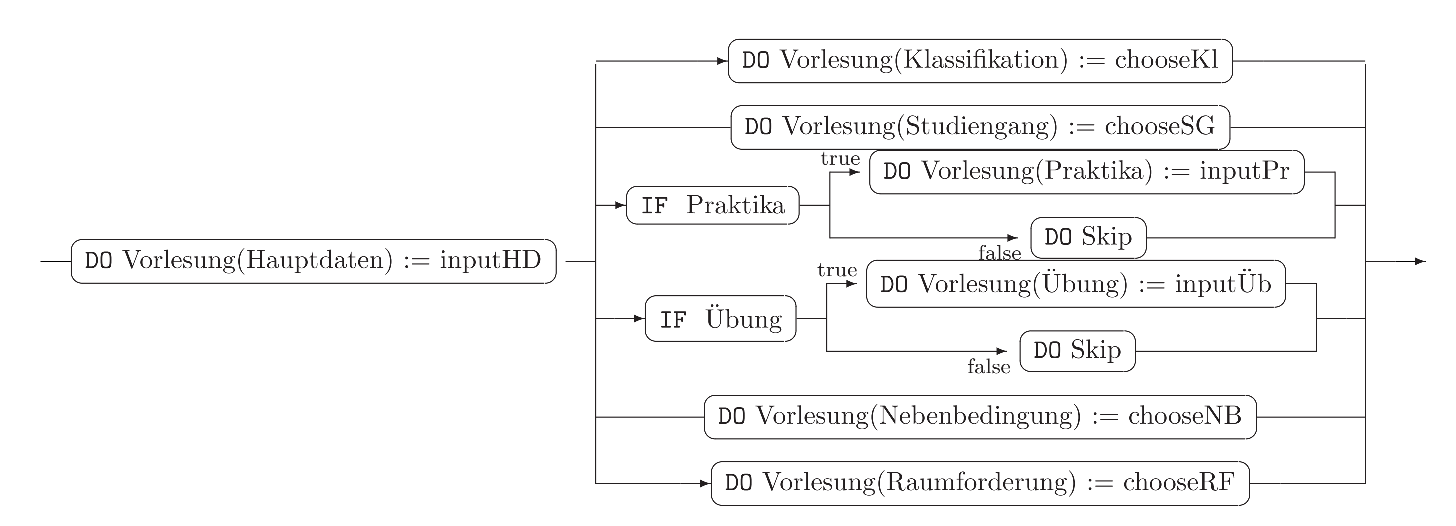
DO $f(s_1, ..., s_n) := t$

Aufruf eines Programmes aus der Programmbibliothek: DO $r(t_1, ..., t_n)$

Es kann ein Programm aus der Programmbibliothek aufgerufen werden.

DO $r(t_1, ..., t_n)$

Mit diesen allgemeinen Transitionen können wir auch komplexere Programme zusammenstellen. Diese Programme basieren auf einer parallelen Ausführung. Das folgende Beispiel stellt die Alternativen zur Eingabe der Hauptdaten zu einem Lehrveranstaltungsangebot vor. Danach können alle erforderlichen sonstigen Daten bereitgestellt werden, wie z.B. Raumforderungen oder auch optionale Informationen, wie z.B. Angaben zu Übungen und Praktika. Raumdaten, Studiengangsdaten, Angaben zu Nebenbedingungen und die Klassifikation der Lehrveranstaltung werden nicht neu erstellt, sondern aus der Datenbank abgefragt und zugeordnet.



Abstrakte Semantik von Datenbank-Transitionen

Das Datenbank-Schema definiert die Strukturierung der Datenbank. Ein Zustand der Datenbank kann durch eine Modifikation partiell geändert werden. Änderungsoperationen $T(s_1, ..., s_n) := t$ vom Teiltyp T' von T basieren auf Anfragen. Sie sind auf einem Objekt einer Klasse T^C definiert, falls $|\, \sigma_{T'=(s_1,...,s_n)}(T^C)\,| \leq 1$ gilt.
Eine Menge $\mathcal{U} = \{T_i(s_{i,1}, ..., s_{i,n_i}) := o_i \mid 1 \leq i \leq m\}$ von objekt-basierten Änderungsoperationen ist *konsistent*, falls aus $T_i(s_{i,1}, ..., s_{i,n_i}) = T_j(s_{j,1}, ..., s_{j,n_j})$ für $1 \leq i < j \leq m$ die Gleichheit $o_i = o_j$ folgt.

Das *Resultat* der Ausführung einer konsistenten Menge $\mathcal{U}$ von Änderungsoperationen führt zu einer Zustandsänderung der Datenbank $\mathcal{ER}^C$ zu $\mathcal{ER}^C + \mathcal{U}$

$$(\mathcal{ER}^C + \mathcal{U})(o) = \begin{cases} Update(T_i, s_{i,1}, ..., s_{i,n_i}, o_i) & \text{falls } T_i(s_{i,1}, ..., s_{i,n_i}) := o_i \in U \\ \mathcal{ER}^C(o) & \text{andernfalls} \end{cases}$$

für Objekte o of $\mathcal{ER}^C$.

Ein parametrisiertes Programm $r(x_1, ..., x_n) = P$ der Stelligkeit n besteht aus einem Programmnamen r, einer Transitionsregel P und einer Menge $\{x_1, ..., x_n\}$ von freien Variablen von P.

Eine Datenbank $\mathcal{ER}^C$ ist ein Modell von ϕ (kurz bezeichnet als $\mathcal{ER}^C \models \phi$) falls $[\![\phi]\!]_\zeta^{\mathcal{ER}^C} = true$ für alle Variablenbelegungen ζ für die freien Variablen von ϕ.

Eine Workflow-Maschine $\mathcal{W} = (\mathcal{ER}, \mathcal{E}R^{C_0}, \mathcal{P}, \Sigma)$ basiert auf einem Datenbank-Schema $\mathcal{ER}$, einer initialen Datenbank $\mathcal{E}R^{C_0}$, einer Menge von parametrisierten Programmen und einem ausgezeichneten Programm, das Hauptprogramm genannt wird, sowie den dynamischen Integritätsbedingungen.

Eine Transitionsregel P führt zu einer Menge $\mathcal{U}$ von Änderungsoperationen in einem Zustand $\mathcal{E}R^C$, falls dieser konsistent ist. Sie verändert den Zustand der Datenbank mit einer Variablenbelegung ζ zu $\texttt{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U})$.

Die Semantik einer Transitionsregel wird durch einen Kalkül mit Regeln der Form

$$\frac{\text{Voraussetzung}_1, ..., \text{Voraussetzung}_n}{\text{Folgerung}} \qquad \text{Bedingung}$$

definiert.

Wir verzichten hier auf die vollständige Angabe der Semantik und verweisen auf die Literatur. Als Beispiel führen wir die folgenden Regeln an, ohne auf den Beweis dieser Regeln einzugehen:

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U}) \ , \ \text{yields}(\mathcal{Q}, \mathcal{E}R^C, \zeta, \mathcal{V})}{\text{yields}(\text{DO } \mathcal{P} \text{ PAR } \mathcal{Q}, \mathcal{E}R^C, \zeta, \mathcal{U} \cup \mathcal{V})} \qquad \mathcal{U} \cup \mathcal{V} \quad \text{konsistent}$$

Die Konsistenzbedingung kann weggelassen werden, wenn man die Theorie der partiell-geordneten Durchläufe für ASM anwendet. Wir wollen jedoch hier nicht im Detail auf die Theorie eingehen.

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta[x \mapsto a], \mathcal{U})}{\text{yields}(\text{LET } x = t \text{ IN } \mathcal{P} \text{ DO } \mathcal{P} \ , \ \mathcal{E}R^C, \zeta, \mathcal{U})} \qquad \text{wobei} \quad a = [\![t]\!]_\zeta^{\mathcal{E}R^C}$$

$$\frac{\forall \, a \in I \ : \ \text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta[x \mapsto a], \mathcal{U}_a)}{\text{yields}(\text{FOR ALL } x \text{ WITH } \phi \text{ DO } \mathcal{P} \ , \ \mathcal{E}R^C, \zeta, \bigcup_{a \in I} \mathcal{U}_a)} \qquad \text{wobei} \quad I = range(x, \phi, \mathcal{E}R^C, \zeta)$$

Der Wertebereich $range(x, \phi, \mathcal{E}R^C, \zeta))$ ist definiert durch die Menge $\{o \in \mathcal{E}R^C \mid [\![\phi]\!]_{\zeta[x \mapsto a]}^{\mathcal{E}R^C} = true\}$.

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta[x \mapsto a], \mathcal{U})}{\text{yields}(\text{CHOOSE } x \text{ WITH } \phi \text{ DO } \mathcal{P} \ , \ \mathcal{E}R^C, \zeta, \mathcal{U})} \qquad \text{wobei} \quad a \in range(x, \phi, \mathcal{E}R^C, \zeta)$$

$$\frac{}{\text{yields}(\text{CHOOSE } x \text{ WITH } \phi \text{ DO } \mathcal{P} \ , \ \mathcal{E}R^C, \zeta, \emptyset)} \qquad \text{falls} \quad range(x, \phi, \mathcal{E}R^C, \zeta) = \emptyset$$

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U}) \ , \ \text{yields}(\mathcal{Q}, \mathcal{E}R^C + \mathcal{U}, \zeta, \mathcal{V})}{\text{yields}(\text{DO } \mathcal{P} \ ; \mathcal{Q} \ , \ \mathcal{E}R^C, \zeta, \mathcal{U} \oplus \mathcal{V})} \qquad \text{falls } \mathcal{U} \text{ konsistent ist}$$

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U})}{\text{yields}(\text{DO } \mathcal{P} \ ; \mathcal{Q} \ , \ \mathcal{E}R^C, \zeta, \mathcal{U})} \qquad \text{falls } \mathcal{U} \text{ inkonsistent ist}$$

$$\frac{}{\text{yields}(\text{SKIP} \ , \ \mathcal{E}R^C, \zeta, \emptyset)}$$

$$\frac{}{\text{yields}(f(s_1, ..., s_n) = t \ , \ \mathcal{E}R^C, \zeta, (\text{Update}(l, v))} \qquad \text{wobei} \quad l = f([\![s_1]\!]_\zeta^{\mathcal{E}R^C}, ..., [\![s_n]\!]_\zeta^{\mathcal{E}R^C}) \quad \text{und} \quad v = [\![t]\!]_\zeta^{\mathcal{E}R^C}$$

$$\frac{\text{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U})}{\text{yields}(\text{IF } \phi \text{ THEN} \mathcal{P} \text{ ELSE } \mathcal{Q} \ , \ \mathcal{E}R^C, \zeta, \mathcal{U})} \qquad \text{falls} \quad [\![\phi]\!]_\zeta^{\mathcal{E}R^C} = true$$

$$\frac{\text{yields}(\mathcal{Q}, \mathcal{E}R^C, \zeta, \mathcal{V})}{\text{yields}(\text{IF } \phi \text{ THEN} \mathcal{P} \text{ ELSE } \mathcal{Q} \ , \ \mathcal{E}R^C, \zeta, \mathcal{V})} \qquad \text{falls} \quad [\![\phi]\!]_\zeta^{\mathcal{E}R^C} = false$$

$$\frac{\text{yields}(\mathcal{P}\frac{t_1, ..., t_n}{x_1, ..., x_n}, \mathcal{E}R^C, \zeta, \mathcal{U})}{\text{yields}(r(t_1, ..., t_N) \ , \ \mathcal{E}R^C, \zeta, \mathcal{U})} \qquad \text{mit} \quad r(t_1, ..., t_n) = P$$

Die angegebene Workflow-Maschine erlaubt eine allgemeine Spezifikation des Verhaltens des Datenbanksystems. Mit dieser Grundlage können wir eine pragmatischere Spezifikationssprache im weiteren verwenden.

Dynamische Integritätsbedingungen

Dynamische Integritätsbedingungen werden in der Literatur meist aufgrund ihrer Kompliziertheit weggelassen. Sie sind jedoch für die Datenbank-Entwicklung nicht minder wichtig. Deshalb führen wir auch einige Klassen explizit ein.

Wir betrachten dazu temporale Klassen vom Typ R:
Jedem potentiellen Objekt o von $dom(R)$ kann eine Lebenszeit $l_R(o) \subseteq \mathbb{N}$ in der Datenbank zugeordnet werden. Damit können wir

- temporale Klassen (R^C, l_R) und
- ihre Schnappschüsse $S(i, R^C, l_R) = \{o \in dom(R) | i \in l_R(o)\}$

einführen.

Gegeben seien eine Formel α über R, eine temporale Klasse (R^C, l_R) über R, und Schnappschüsse $S(0, R^C, l_R)$, ...,$S(i, R^C, l_R)$, ... $S(maxT, R^C, l_R)$.
Wir können nun eine Erfüllbarkeit von Formeln analog zur Modallogik definieren.
Ein *Zeitrahmen* ZR besteht aus einem Paar (TS, W) von Intervallen von $\mathbb{N}$ und einer binären Relation W über TS. Wir bezeichnen mit $maxTS$ den maximalen Zeitpunkt von TS und mit $minTS$ den minimalen Zeitpunkt. Der einfachste Zeitrahmen ist das Intervall $TS = [0, maxTS]$ betrachtet. Die binäre Relation W stellt eine Erreichbarkeit von Intervallen untereinander her. Wir sind damit in der Lage, Zeiträume zu betrachten und ggf. auch voneinander zu separieren.

- Die Gültigkeit von α in einem Schnappschuß $S(i, R^C, l_R)$ ist induktiv wie für statische Integritätsbedingungen definiert und wird mit $(R^C, l_R, i) \models \alpha$ notiert.

- Die Formel α is *notwendig gültig* in (R^C, l_R), zu einem Zeitpunkt $i \in I, I \in TS$ und für einen Zeitrahmen ZR falls $(R^C, l_R, i') \models \alpha$ für alle Intervalle I, I' mit $(I, I') \in W$ und alle Zeitpunkte $i' \in I \cup I'$.
  Wir notieren dies mit $(R^C, l_R, i, ZR) \models \Box\alpha$ bzw. $(R^C, l_R, I, ZR) \models \Box\alpha$
  Die Formel ist gültig in jedem Zeitpunkt des Intervalls I, dem i angehört, und in jedem Zeitpunkt, der durch W aus I erreicht werden kann. In der Modallogik wird zwischen der Gültigkeit von α in I und zu jedem Nachfolgeintervall unterschieden. Wir benötigen diese strikte Unterscheidung nicht. Wir können mit $(R^C, l_R, I, ZR) \models \Box\alpha$ die Gültigkeit ab einer Phase I für alle Folgephasen I' modellieren.

- Eine Formel α ist *notwendig gültig* in (R^C, l_R) und ZR ab I_1 bis zu I_2 für $I_1, I_2 \in TS$ falls $(R^C, l_R, i) \models \alpha$ für alle Intervalle I' mit $(I_1, I') \in W$ bzw. $(I', I_2) \in W$ und $i \in I_1 \cup I_2 \cup I'$.
  Wir bezeichnen die zeitweilige volle Gültigkeit mit $(R^C, l_R, [I_1, I_2], ZR) \models \Box\alpha$.
  Wir können damit die Gültigkeit zwischen Phasen definieren.

- Die Formel α ist *gültig* in (R^C, l_R) und ZR falls $(R^C, l_R, i) \models \alpha$ für jeden Zeitpunkt i jedes Intervalls I des Zeitrahmens (bezeichnet mit $(R^C, l_R, ZR) \models \alpha$).
  In diesem Fall ist α eine statische Integritätsbedingung, falls $\bigcup_{I \in TS} I = [minTS, maxTS]$.

- Die Formel α ist *möglich gültig* in (R^C, l_R) und ZR falls für ein $i \in \bigcup_{I \in TS} I$ $(R^C, l_R, i) \models \alpha$ (bezeichnet mit $(R^C, l_R, ZR) \models \Diamond\alpha$).

Besitzt ein Zeitrahmen ZR Unterbrechungen, dann wird für die Formel α keine Forderung erhoben.

Ein Zeitrahmen wird für die Implementationsschicht direkt durch Phasen repräsentiert. Damit kann die Gültigkeit von Formeln und die Zulässigkeit von Prozessen zu gewissen Zeitpunkten direkt modelliert werden.

Wir können damit auch unterschiedliche Klassen von dynamischen Integritätsbedingungen einführen. Dafür werden der Zeitrahmen $ZR_{\text{Schritt}} = (\{\{i\} | i \in \mathbb{N}\}, \{(\{i\}, \{(i+1)\}) | i \in \mathbb{N}\})$ und $ZR = (\{\{i\} | i \in \mathbb{N}\}, \emptyset)$ sowie $ZR = (\mathbb{N}, \mathbb{N} \times \mathbb{N})$ eingeführt.

Transitionsbedingung: Eine Formel α heißt Transitionsbedingung, falls α notwendig gültig in allen Intervalle von ZR_{Schritt} ist.
Wir notieren Transitionsbedingungen auch durch $\alpha \stackrel{\text{next}}{\longrightarrow} \alpha$.

Allgemeine Vor- und Nachbedingung: Ein Paar von Formeln α und β heißt Vor- und Nachbedingungen falls aus $(R^C, l_R, i, ZR_{\text{Punkt}}) \models \Box\alpha$ die Gültigkeit von $(R^C, l_R, i+1, ZR_{\text{Punkt}}) \models \Box\beta$ folgt.
Wir notieren allgemeine Vor- und Nachbedingungen auch durch $\alpha \stackrel{\text{next}}{\longrightarrow} \beta$.
Wird der Zeitrahmen durch die Anwendung eines Prozesses P oder Programmes P definiert, dann schreiben wir $\alpha \stackrel{P}{\longrightarrow} \beta$.

Temporale Formeln sind mit $(R^C, l_R, i, ZR_{\text{Voll}}) \models \Box\beta$ bzw. $(R^C, l_R, i, ZR_{\text{Voll}}) \models \Diamond\beta$ im Sinne der Modallogik notwendig oder möglich gültig.

In analoger Form können damit auch allgemeine Gültigkeiten temporaler Formeln eingeführt werden:

$\forall_f$: immer (in der Zukunft)

$\forall_p$: immer (in der Vergangenheit)

$\exists_f$: einmal (in der Zukunft)

$\exists_p$: einmal (in der Vergangenheit).

$\mathsf{U}(\alpha, \beta)$: α ist gültig bis β gültig wird.

Weiche (deontische) Integritätsbedingungen werden für ZR_{Schrit} eingeführt:

Obligation: Eine Obligation $O\alpha$ ist durch die Gültigkeit von $(R^C, l_R, 1, ZR_{\text{Schritt}}) \models \Box\alpha$ definiert.

Erlaubnis: Eine Erlaubnis $P\alpha$ ist durch die Gültigkeit von $(R^C, l_R, 1, ZR_{\text{Schritt}}) \models \Diamond\alpha$ definiert.

Verbot: Ein Verbot $F\alpha$ ist durch die Gültigkeit von $(R^C, l_R, 1, ZR_{\text{Schritt}}) \models \Box\neg\alpha$ definiert.

Wir können daraus direkt einige Ableitungsregeln ableiten:

KD0 : Jede Formel der HERM-Logik ist eine deontische Formel.

KD1 : $O(\alpha \rightarrow \beta) \rightarrow (O\alpha \rightarrow O\beta)$

KD2 : $O\alpha \rightarrow P\alpha$ Obligationen umfassen Erlaubnisse.

KD3 : $P\alpha \leftrightarrow \neg O \neg \alpha$ Die Erlaubnis ist dual zur Obligation.

KD4 : $F\alpha \leftrightarrow \neg P\alpha$ Verboten heißt "nicht erlaubt".

Weitere allgemeingültige Formeln der deontischen Logik sind z.B.:

- $O\alpha \leftrightarrow \neg P \neg \alpha$
- $O(\alpha \wedge \beta) \leftrightarrow O\alpha \wedge O\beta$
- $\neg(O\alpha \wedge O\neg\alpha)$
- $P(\alpha \vee \beta) \leftrightarrow P\alpha \vee P\beta$
- $(O\alpha \vee O\beta) \rightarrow O(\alpha \vee \beta)$
- $O\alpha \rightarrow O(\alpha \vee \beta)$
- $P(\alpha \wedge \beta) \rightarrow P\alpha \wedge P\beta$
- $F\alpha \rightarrow F(\alpha \wedge \beta)$
- $(O\alpha \wedge P\beta) \rightarrow P(\alpha \wedge \beta)$
- $(O\alpha \wedge O(\alpha \rightarrow \beta)) \rightarrow O\beta$

- $(P\alpha \wedge O(\alpha \rightarrow \beta)) \;\rightarrow\; P\beta$
- $(F\beta \wedge O(\alpha \rightarrow \beta)) \;\rightarrow\; F\alpha$
- $F\beta \wedge Fr \wedge O(\alpha \rightarrow (\beta \vee \gamma)) \;\rightarrow\; F\alpha$
- $\neg(O(\alpha \vee \beta) \wedge F\alpha \wedge F\beta)$
- $(O\alpha \wedge O((\alpha \wedge \beta) \rightarrow \gamma)) \;\rightarrow\; O(\beta \rightarrow \gamma)$
- $O(\neg\alpha \rightarrow \alpha) \;\rightarrow\; O\alpha$
- $O\beta \;\rightarrow\; O(\alpha \rightarrow \beta)$
- $F\alpha \;\rightarrow\; O(\alpha \rightarrow \beta)$
- $O\neg\alpha \;\rightarrow\; O(\alpha \rightarrow \beta)$
- $\neg\alpha \;\rightarrow\; (\alpha \rightarrow O\beta)$
- $\neg O(\alpha \wedge \neg\alpha)$
- $(O\alpha \wedge O(\alpha \rightarrow \beta) \wedge (\neg\alpha \rightarrow O\neg\beta) \wedge \neg\alpha) \;\leftrightarrow\; \text{false}$
- $\alpha \rightarrow \beta \;/\; O\alpha \rightarrow O\beta$

Wir werden uns jedoch im weiteren auf Transitionsbedingungen und Vor- und Nachbedingungen konzentrieren, sowie auf weiche Integritätsbedingungen der deontischen Logik.

Unterscheidung von Handlungsabläufen für Funktionalität und Interaktivität

In klassischen Ansätzen werden Handlungsabläufe zur Spezifikation der Funktionalität und zur Spezifikation der Interaktivität auf gleiche Art und Weise durch Workflows dargestellt. Diese Darstellung ist aufgrund einer ganzen Reihe von Gründen irreführend und führt zu einem *Workflow-Impedance-Mismatch*:

- Workflows zur Spezifikation der Funktionalität umfassen auch Prozesse der Systeme, die auf dem Niveau der Interaktivität keine Rolle spielen. Deshalb sind Workflows *überladen*.
- Handlungsabläufe der Realität müssen sich nicht zwingend im Workflow widerspiegeln. Demzufolge werden Workflows *funktional unvollständig*.
- Handlungsabläufe auf Systemniveau und auf Interaktionsniveau unterscheiden sich im Abstraktionsniveau. Deshalb besitzen sie eine unterschiedliche *Granularität*.
- Handlungsabläufe auf Interaktionsniveau stellen auch die Zusammenarbeit von Akteuren dar, die sich nicht zwingend im System widerspiegeln muß. Demzufolge sind Workflows *organisatorisch unvollständig*.
- Workflows zur Spezifikation der Funktionalität sollten den konkreten Handlungsablauf nicht in Beziehung zum Kontext setzen. In klassischen Herangehensweisen werden aber die unterschiedlichsten Varianten des gleichen Workflows je nach Kontext als eigenständiger Workflow dargestellt. Es entsteht eine *Workflow-Lawine*, deren Beherrschung und Überschaubarkeit nicht mehr gegeben ist.

Wir bevorzugen dagegen eine Trennung von dynamischen Gesichtspunkten in

Stories zur Darstellung der Handlungsabläufe auf Interaktionsniveau und

Workflows zur Darstellung der Handlungsabläufe auf Systemniveau.

Workflow-Klassen, Workflows und Workflow-Felder

Die Beschreibung der Handlungsabläufe lehnen wir dabei an die Formenlehre an. In der Morphologie kann ein Wort in allen seinen Variationen dargestellt werden durch

eine Stammform zur Parametrisierung der unterschiedlichen Dimensionen, wie z.B. Zeitdimension und Akteurdimension,

die Wortbildung, d.h. durch Regeln zur Assoziation von Wörtern zu komplexeren Ausdrücken wie z.B. Ableitung, Zusammensetzung und Abkürzung, und

die Flexion zur Ableitung von Varianten und zur Erfassung von Ausnahmen.

Ein morphologisches Merkmal erlaubt die Kennzeichnung der Ableitungsdimensionen eines Begriffes je nach seiner Kategorie (Verb, Nomen, Artikel/Pronomen, Adjektiv, Partikel) durch

Deklination der drei Merkmale

- *Kasus*, mit dem eine Assoziierung der Worte zu thematischen Rollen und der Art der Assoziierung (Nominativ ('wer', 'was'), Akkusativ ('wen', 'wohin', 'wie lange'), Genitiv ('wessen'), Dativ ('wem', 'für wen'), Ablativ ('wodurch', 'womit', 'von wem', 'weswegen', 'wann') und Vokativ (zur direkten Anrede)) determiniert wird,
- *Genus*, mit dem eine Kategorisierung z.B. zum Geschlecht (Maskulinum, Femininum, Neutrum) vorgenommen wird, und
- *Numerus*, mit dem eine Einzelbehandlung oder eine Gruppenbehandlung ermöglicht wird,

Konjugation zur Instantiierung von n-wertigen (-valenten) Beziehungen mit

- *Numerus* zur Assoziierung mit Kardinalität (Singular (card(R,E) = (0,1)), Dual (card(R,E) = (0,2)) bzw. Plural (card(R,E) = (0,n))) ,
- *Personalformen* zur Ausrichtung der Beziehung ('ich' = $\rightarrow$, 'er' = $\curvearrowright$, 'wir' = $\leftrightarrow$),
- *Tempus* zur zeitlichen Relativierung (Präsenz, Perfekt, Plusquamperfekt, etc.),
- *Modus* zur Bewertung der Modalität (Indikativ (als Feststellung z.B. durch Teilklasse), Imperativ ((1,1) bzw. (1,n)), Konjunktiv I (zur Darstellung der allgemeinen Möglichkeit bzw. Wunsches), Konjunktiv II (zur Abgrenzung einer spekulativen Möglichkeit)) und
- *Genus verbi* zur Darstellung der Beziehungsform (aktiv bzw. passiv)

oder

Komparation zur Darstellung von Steigerungsformen

- Positiv bzw. einfach positiv,
- Komparativ bzw. Vergleichstufe bzw. Höherstufe und
- Superlativ als Höchststufe sowie
- Elativ als absoluter Superlativ

und Ausprägungen des Wortes.

Da wir die Theorie der Wortfelder [Kun92] zu Konzeptfeldern [DT04] bzw. Konzeptrahmen erweitern, wird für ein Konzeptfeld eine Kontextualisierung (oder Konjugation) durch Instantiierung der Parameter

Akteursprofile und -portfolio,

Wiederholungsprofil,

Zeitprofil,

deontischer Modus mit imperativen, konjunktiven und indikativen Ausprägungen und

Aktionsform und Handlungsrichtung zur Darstellung der Beziehung zwischen Akteur und Handlungsablauf

erreicht. Damit werden insbesondere die Parameter der Stammform des Konzeptfeldes durch entsprechende Werte angepaßt. Ein Konzeptfeld ist ein generisches Konzept, aus dem ein Konzept durch Instantiierung einer Reihe von Merkmalen abgeleitet wird. Dieser Zugang entspricht in der Theorie der Wortfelder der Komponentenanalyse. Wir verwenden diese Ableitungsbeziehung analog zu den Erkenntnissen der Sprachwissenschaft. Wir können z.B. aus dem Konzeptfeld *Lebewesen* wie folgt Konzepte ableiten:

Lebewesen	Belebtheit	Kategorie=*Mensch*	Geschlecht=*weiblich*	Lebensalter=*Erwachsen*	...
Mann	+	+	-	+	...
Mädchen	+	+	+	-	...
Rüde	+	-	-	+	...
Welpe	+	-	+/-	-	...

Analog kann auch eine generelle Klasse eingeführt werden.

Diese Beobachtung führte V.J. Propps [Pro72] zu seiner Spezifikation der Märchen. Er stellte z.B. für das Märchen *Die wilden Schwäne* den Ausdruck

$$ib^1a^1c^1A^1B^4C \nearrow \{Sch^1H^1_\neg Z^1_\neg \| sch^7H^7Z^9\}W^4L^1 \searrow V^1[Sch^1H^1Z^9 \equiv R^4] \times 3$$

auf. Die Buchstaben werden jeweils für eine Konzeptfeld verwandt, z.B. I für das Eröffnungsfeld, a für Ortsbewegungen, b für Verbote, c für Verletzungen der Verbote, A für Schädigungen, C für eine einsetzende Gegenhandlung, $\nearrow$ und $\searrow$ für Ortsveränderungen, Sch für Schenkungen, H für Reaktionen des Helden, Z für den Empfang eines Zaubermittels, $\|$ für eine Parallelhandlung, W für eine Wegweisung, L für die Aufhebung des Unglücks, V für die Verfolgung und R für die Rettung. Die einzelnen Schritte können durch Annotation auch verfeinert oder negiert werden. Außerdem kann eine Wiederholung angezeigt werden, z.B. die dreimalige Wiederholung durch 3.

Wir unterschieden in Anlehnung an die Theorie der Konzeptfelder zwischen

Workflow-Klassen, in denen Workflows als Einzelelemente enthalten sind,

einem **Workflow** als Objekt einer Workflow-Klasse und

Workflow-Feldern, mit denen ein Rahmen der Workflow-Klasse angegeben werden kann.

Ein Typ einer Workflow-Klasse kann aus einer oder mehreren Stammformen bestehen.

Ein Workflow-Feld besteht aus

einer Menge von Stammformen,

einer Menge von dynamischen Integritätsbedingungen denen die Workflows eines Feldes genügen müssen,

einer Menge von Bildungsformen zur Assoziation mit anderen Workflow-Feldern und

einer Menge von Flexionen zur Ableitung von Workflows aus dem Workflow-Feld.

Wir nehmen oft vereinfachend an, daß ein Workflow-Feld nur eine einzige Stammform besitzt. und daß eine Workflow-Klasse nur Workflows eines Workflow-Feldes enthält. Sie muß nicht alle möglichen Workflows dieses Feldes enthalten, sondern kann auch nur einige (aktuelle) Workflows enthalten.

Diese Unterscheidung wurde in unserer Arbeit erstmals für eine e-Learning-Site konzipiert. Diese Site erlaubt eine Entfaltung einer Lerneinheit je nach Meta-Information, Handlungs-, Akteurs- und Datenkontexten sowie der Handlungsvorgeschichte. Damit kann ein Lernfeld als allgemeine Lerneinheit angesehen werden, bei der

die Stammform als Ausdruck über Lernelementen gegeben ist,

die durch Ableitungsregeln zu einem komplexen Lernmodul erweitert wird, so dass ein Lernender auch seine entsprechenden Lernelemente angeboten bekommt, und

durch Flexion die Variantenvielfalt sowie die Ausnahmen auffaltbar sind. Flexionsregeln erlauben eine Erweiterung

mit dem Akteursprofil und -portfolio,

mit dem Wiederholungsprofil,

mit dem Zeitprofil,

mit dem deontischen Modus und

mit den Aktionsformen und der Handlungsrichtung.

Diese Erweiterung wurde bereits in einigen Arbeiten, die Workflow-Spezifikationen aus der Sicht der Praxis kritisierten, gefordert. Es wurde insbesondere beobachtet,

- daß ein Arbeitsablauf hoch-parallel ist,
- daß ein Arbeitsablauf eine Rückkopplung mit Wartezeiten erfordert und
- daß die Organisation der Arbeit oft fremdgesteuert ist.

Wir verallgemeinern diese Formenlehre von Handlungssträngen und führen dazu allgemeine Workflow-Felder ein:

Das Eröffnungsfeld ist gekennzeichnet durch

- die Darstellung des Kontextes, der bei Assoziation des Workflow-Feldes mit anderen Feldern den Kontext dieser Felder dominiert,
- die Darstellung der Akteure,
- die Darstellung der Situation und
- die Assoziation mit Sichtentypen sowohl für die Input- als auch die Retrieval- als auch die Outputdaten.

Das Ausgangsfeld dient zur Meta-Spezifikation und erlaubt außerdem noch eine Einbettung der räumlichen und zeitlichen Rahmenbedingungen sowie auch der Motivation und der Ursachen.

Das Handlungsschrittfeld wird spezifiziert durch

- die Angabe der Verbindungsparameter,
- die Angabe der Begleitelemente und Kontextbedingungen,
- der Rollen der Akteure und
- Sichtentypen.

Das Übergabefeld erlaubt den Übergang von Objekten einer Sicht eines Akteurs auf Objekte einer Sicht eines anderen Akteurs. Zusätzlich kann der Kontext und auch der Vertrag des Überganges spezifiziert werden.

Das Arbeitsfeld erlaubt die Bearbeitung von Daten über den Sichtentypen und deren Versand an andere Akteure bzw. deren Einbringen in das System.

Neben diesen Basisfeldern können wir auch Konstruktionsfelder spezifizieren, mit denen Felder kombiniert werden können:

Das Verzweigungsfeld unterstützt eine Verzweigung von Workflows in synchronisierte Workflows, die parallel unter Einhaltung der Synchronisationsbedingungen ablaufen können.

Das Wiederholungsfeld stellt den Rahmen für eine wiederholte Ausführung eines Workflows.

Das Bulk-Feld ist an Parameter gebunden, für die das Workflow-Feld insgesamt abgearbeitet wird.

Wir haben diese Theorie der Workflow-Felder mit den Kompositionsoperationen für Workflows harmonisiert, damit wird eine entsprechende Entfaltung der komplexen Workflow-Felder vornehmen können.

Generische Workflows und entfaltbare Workflows

Workflow-Felder erlauben oft eine einfache Darstellung durch entsprechende Ausdrücke. Können Workflow-Felder durch eine Stammform spezifiziert werden, dann nennen wir diese Stammform generischer Workflow. Generische Workflows stellen ein Analogon zu generischen Operationen wie *insert*, *delete* und *update* dar, bei denen eine Instantiierung durch Angabe der Strukturen der Typen erfolgt, für deren Klassen sie angewandt werden. Ebenso wie generische Operationen können generische Workflows durch Instantiierung in konkrete Workflows überführt werden. Die Parameter können auch abhängig voneinander sein. Wir unterscheiden hierbei die folgenden speziellen Typen:

Entfaltbare Workflows sind generische Workflows mit einem generischem Laufzeit-Workflow, bei denen die instantiierbaren Parameter keine Nebenbedingungen auf andere Parameter besitzen. Sie können zur Laufzeit voll entfaltet werden. Typische entfaltbare Workflows sind Workflows für Gruppenarbeitsplätze, die jedem Mitglied die gleiche Arbeitsplattform bieten.

Parallelisierte Workflows sind generische Workflows, bei denen ein Zwischenstand und To-Do-Listen mitgeführt werden und zur Laufzeit mit entsprechenden Werten belegt werden können, die zu anderen Workflows Beziehungen besitzen z.B. durch Ressourcen-Sharing und gemeinsam mit diesen ausgeführt werden können.

Multiple-choice Workflows sind generische Workflows, die Varianten für Rollen, für die freie Auswahl von Daten und die Bündelung mit anderen Workflows bereitstellen.

Transaktions-basierte Meta-Workflows sind generische Workflows, deren Ausführungsmodell eine Ressourcen- und Rollenverwaltung einschließt, die auch über Rücknahme- oder Kompensationsteilfelder verfügen und deshalb einer Transaktionssemantik unterliegen.

Ein entfalteter Workflow ist ein vollständig instantiierter Workflow. Alle Parameter eines entfaltbaren Workflow sind mit entsprechenden Werten belegt. In Bild 30 wird die Beziehung zwischen einem generischen Workflow und einem entfalteten Workflow dargestellt. Ein entfalteter Workflow ist demzufolge ein Durchlauf oder eine spezielle Instanz eines generischen Workflows.

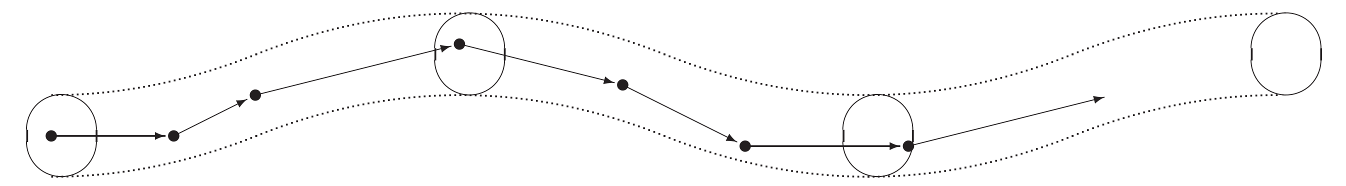


Bild 30: Generische und entfaltete Workflows

Komposition von Workflow-Feldern zu Programmen

Auf Seite 68 wurde bereits die Workflow-Maschine eingeführt. Oft erscheint es einfacher, eine algebraische Notation mit abgeleiteten Operationen zu verwenden. Obwohl die Workflow-Maschine zur Komposition der Workflow-Felder ausreicht, wollen wir im Abschluß noch eine algebraische Sprache anführen. Diese Sprache harmonisiert mit der Algebra, die wir SiteLang verwenden:

Ein atomares Workflow-Programm ist ein Workflow-Feld.

Einfache Steueranweisungen sind

- **die sequentielle Ausführung ; ,** bei der Workflow-Programme sequentiell nacheinander ausgeführt werden, wobei die Semantik des ersten Programmes die Semantik des zweiten Programmes ergänzt und das leere Programm entsteht, wenn die Vereinigung der Semantik zum Widerspruch führt.

parallele Verzweigung $|^{\wedge}|$, bei der Programme parallel ausgeführt werden können und das terminiert, wenn beide Programme terminieren,

exklusive Auswahl $|^{\vee}|$, bei der genau ein Programm zur Ausführung nichtdetermistisch ausgewählt werden kann,

Synchronisation $|^{sync}|$, die eine parallele Ausführung mit einer Synchronisationsbedingung zuläßt, und

einfaches Mischen $+$, bei dem zwei alternative Programme verbunden werden können.

Erweiterte Verzweigungs- und Synchronisationsanweisungen sind

mehrfache Auswahl , bei der verschiedene Ausführungspfade gewählt werden können,

mehrfaches Mischen , bei dem verschiedene Ausführungspfade gemischt werden können,

Diskriminator, bei dem verschiedene Ausführungspfade ohne Synchronisation gemischt werden können, wobei Teilprogramme nur einmal ausgeführt werden,

n-out-of-m.Verbund , bei dem verschiedene Ausführungspfade mit partieller Synchronisation gemischt werden können, wobei Teilprogramme nur einmal ausgeführt werden, und

synchronisierter Verbund, bei dem verschiedene Ausführungspfade mit vollständiger Synchronisation gemischt werden können, wobei Teilprogramme nur einmal ausgeführt werden.

Strukturelle Steueranweisungen sind

Wiederholung $*$, bei der Programme beliebig oft ausgeführt werden können und

implizite Termination$\downarrow$, die eine Beendigung des Programmes hervorruft.

Datenabhängige Steueranweisungen sind

statische Steueranweisungen , deren Steuerung mit Bedingungen erfolgt, die bereits zur Compilezeit geprüft werden können,

statische Steueranweisungen, deren Steuerung mit Bedingungen erfolgt, die erst zur Laufzeit geprüft werden können,

Steueranweisungen mit A-priori-Laufzeitannahmen erlauben eine Voreinstellung durch Erzeugung von einer beschränkten Menge von Wiederholungen und

Steueranweisungen mit Synchronisationsbedingen, bei denen beliebig viele Alternativen parallel ausgeführt werden können und eine Synchronisation bei Abschluß erfolgt.

Zustandsbasierte Steueranweisungen sind

die verzögerte Auswahl, bei der alle Alternativen ausgeführt werden und eine Auswahl der Alternative erst nach Ausführung erfolgt,

die verbundene parallele Ausführung, bei der die Alternativen in zufälliger Reihenfolge sequentiell ausgeführt werden, und

die meilenstein-basierte Steuerung, bei der eine Aktivität ausgeführt wird, bis ein Meilenstein erreicht ist.

Abbruchanweisungen sind

Abbruchaktion , bei der eine Aktion abgebrochen wird, und

Fallabbruch , bei der ein Fall abgebrochen wird.

Diese Algebra kann durch die Algebra der Workflow-Maschine ausgedrückt werden. Wir verstehen sie deshalb eher als “syntaktischen Zucker”, der die Spezifikation von Workflow-Programmen vereinfacht.

Mit dieser Algebra führen wir eine rigide Klammerung ein. Damit sind nicht mehr alle Ausdrücke

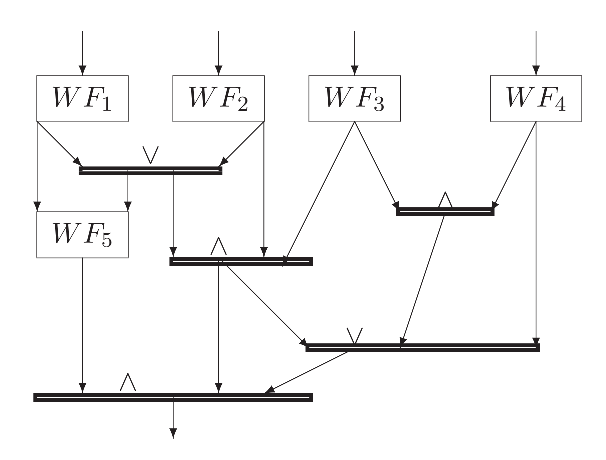


Bild 31: Ein überlagertes und verwirrendes Workflow-Programm

die Auseinandersetzung mit kondensierten und überlagerten Programmen wie in Bild 31 dargestellt. Da die Programmierung von einer klaren Semantik profitiert, erlauben wir diese Art von Konfusion nicht bei der Spezifikation.

Dieses Programm vermischt Ausführung, Sequentialität und Nebenbedingungen zur Entscheidung. Die Alternativen sind vereinfachbar, wenn sicher ist, daß z.B. WF_2 vor WF_1 terminiert und mit den Resultaten von WF_1 übereinstimmt, dann kann WF_5 ohne Berücksichtigung von WF_2 nur auf WF_1 aufbauen.

Das Programm in Bild 31 besitzt gleichzeitig auch die klassische AND-OR-Falle. Analog kann auch die OR-AND-Falle spezifiziert werden. Beide "Fallen" sind in Bild 32 illustriert.

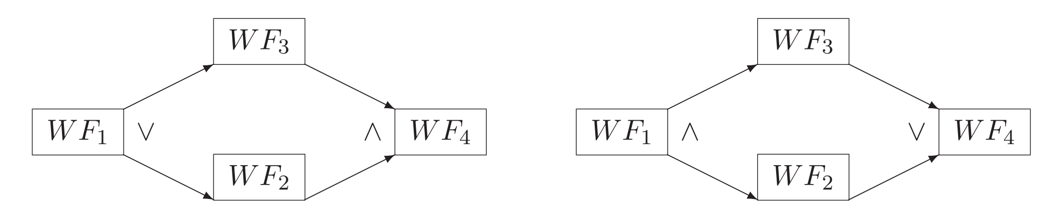


Bild 32: Ein OR-AND- und ein AND-OR-Workflow-Programm

Diese Fallen sind relativ leicht aufzulösen, wenn man die Resultatssemantik betrachtet. In diesem Falle sind beide Programme durch AND-AND-Programme repräsentiert. Betrachtet man dagegen die Ausführungssemantik, dann klaffen die beiden Programme auseinander.

Noch schwieriger sind Workflow-Semantiken, bei denen eine Synchronisation sowohl am Ende als auch zu Beginn einer Verzweigung erfolgen kann. In diesem Fall erhält auch die Verzweigung eine andere Semantik.

Aus diesen Gründen bevorzugen wir die etwas rigidere Semantik der Workflow-Maschine mit der Semantik der abstract state machine. Sie hat zugleich auch den Vorteil, eine Verfeinerung zuzulassen und auch Konflikte auf Datenebene durch Zurücknahme auflösen zu können.

6 Sprachen zur Darstellung der Sichtensuite

Zwei Seelen wohnen, ach! in meiner Brust.
Die eine will sich von der andern trennen;
Die eine hält mit derber Liebeslust
Sich an die Welt mit klammernden Organen;
Die andre hebt gewaltsam sich von Dust
Zu den Gefilden hoher Ahnen.
Goethe, Faust, Erster Teil, Vor dem Thor, Faust

Die Sichtenspezifikation kann analog zur Spezifikation der Strukturierung vorgenommen werden. Sie basiert außerdem auf den Informationen über die Dialoge. Wir unterscheiden drei Arten von Sichten:

Arbeitssichten, mit denen eine Bearbeitung von Daten, das Retrieval von Daten und die Ein- und Ausgabe von Daten auf dem Datenbankschema aufsetzend ermöglicht wird,

Sichten zur Interaktion von Systemen, die zur Unterstützung von verteilten, föderierten oder interoperablen Systemen erstellt werden, und

Sicherungssichten, mit denen die Zugriffs- und Modifikationssicherung für die Datenbank erfolgen kann.

Sicherungssichten werden während der Spezifikation von Sicherheitsanforderungen eingeführt und interessieren uns hier nicht vordergründig. DBMS-Interaktionssichten werden in der konzeptionellen und der Implementationsschicht auf der Grundlage von Verteilungskonzepten entwickelt. Sie werden dort mit betrachtet.

Die Spezifikation der Sichten kann auch in die Spezifikation der Schemata mit eingebettet werden. Da für die Akteure Daten nur im Rahmen der Dialoge von Interesse sind und diese Dialoge auch spezifisch aufbereitete Daten erfordern, ist eine explizite Modellierung der Sichten angebracht.

Wir wollen außerdem eine Spezifikation der Anwendung auf der Grundlage eines sichtenorientierten Zuganges unterstützen. Deshalb benötigen wir explizite Konzepte zur Darstellung des *Zusammenhanges von Sichten*. Dieses Konzept der Sichtenkooperation wird deshalb in diesem Abschnitt ebenfalls eingeführt. Der sichtenorientierte Entwurf konzentriert sich stärker auf die Spezifikation der Aspekte der Anwendung, die mehr den Anwender betreffen. Es wird angenommen, daß eine Integration der einzelnen Sichten - so wie dies für die Anwendung eigentlich der Fall ist - eine lösbare Aufgabe ist. Das steht im Widerspruch zu theoretischen Resultaten. Die *Sichtenintegration* ist eine *algorithmisch unlösbare* Aufgabe. Es existiert *kein Algorithmus*, der entscheidet, ob zwei Sichten integriert werden können. Das Sichtenintegrationsproblem ist auch *nicht semientscheidbar*, d.h. es existiert auch kein Algorithmus, der für Sichten, die integriert werden können, die integrierende Sicht berechnet. Aus diesen Resultaten kann man schließen, daß ein sichtenorientierter Entwurf nicht möglich ist. Wird aber eine konkrete Anwendung betrachtet, dann erscheint auch in vielen Fällen eine Kombinierbarkeit der unterschiedlichen Aspekte der Anwendung gegeben. Wir pflegen deshalb das Wissen um die Integration der Daten direkt im Sichtenintegrationsschema.

Wir unterscheiden *Sichten für strukturelle Aspekte* und *Sichten für funktionale Aspekte*. Diese unterschiedlichen Begriffe wollen wir im weiteren auseinanderhalten.

Ein ***sichtenorientierter struktureller Entwurf*** ist am einfachsten in der Top-Down-Strategie integrierbar. In diesem Fall wird zur Darstellung der strukturellen Zusammenhänge ein *Skelett* benutzt. Es dient zur expliziten Spezifikation der Abbildungen von einzelnen Konstrukten der Sichten untereinander. Jeder neue Entwurfsschritt, der sich auf eine andere Sicht aufgrund dieses Skeletts auswirken kann, zieht eine *Entwurfsobligation* nach sich. Entwurfsobligationen können sofort nach einem Schritt betrachtet werden oder im *deferred*-Modus auch zu einem späteren Zeitpunkt bearbeitet werden. Der späteste praktikable Zeitpunkt ist das Entstehen weiterer Obligationen aus diesen Obligationen. In diesem Fall treten typische Probleme der Sichtenintegration wie die im folgenden behandelten Probleme nicht auf.

Ein ***sichtenzentrierter funktionaler Entwurf*** orientiert sich an den Hauptprozessen und den Dialogen. Es wird für jeden Prozeß bzw. Dialog eine entsprechende Sicht erzeugt, die die Verarbeitung der Daten ermöglicht. Daten können unterschieden werden in *Retrievaldaten*, die mit einer Retrievalanweisung anhand der Datenbank gewonnen werden, in *Inputdaten*, die ein Benutzer in eine Datenbank einfügt, *Outputdaten*, die einem anderen Prozeß (z.B. einem Outputprozeß) übermittelt werden, und *Begleitdaten*, die in einem Prozeß als Zusatzinformation dienen bzw. von anderen Prozessen stammen. Diese Daten können zusätzlich *Displaydaten* sein.

Für die Entwicklung von Informationssystemen konzentrieren wir uns auf eine Datenbanklösung. Deshalb hat der strukturelle Entwurf einen höheren Stellenwert als der funktionale Entwurf. Die Unterscheidung der Daten aus dem funktionalen Entwurf behalten wir jedoch bei.

Sichten im Abstraktionsschichtenmodell

Wir können, wie in Bild 33 dargestellt, auch für die Sichtenspezifikation das Abstraktionsschichtenmodell verwenden. Da die Sichten aber eine Hilfskonstruktion sind und in engem Zusammenhang zum Schema und zu den Dialogen stehen, ist eine isolierte Modellierung der Sichten nicht sinnvoll. Im einzelnen verwenden wir die folgenden Schritte:

Sichten des Lastenhefts: Mit der strategischen Informationsanalyse erhalten wir Informationen zu den unterschiedlichen Ansichten der Akteure zur Datenbank. Diese Ansichten können im Nachgang zum Struktur-, Funktions- und Dialogentwurf zur Entwicklung einer Vorstellung zu den einzelnen Sichten genutzt werden. Es wird eine Produktdatenskizze mit einer Grobstrukturierung der Produktdaten entwickelt. Diese Produktdatenskizze ist mit der Konzeptlandkarte, dem Diskurs und der Produktfunktionalität abzugleichen. Zur Darstellung der Produktdaten wird ein allgemeines HERM-Diagramm mit den Haupttypen entwickelt.

Sichten des Pflichtenhefts: Es wird eine Sichtenskizze entwickelt. Jede dieser Sichtenskizzen basiert auf Begriffen der Anwendung. Wir nennen die Darstellung dieser Begriffe *ontologische Einheit*. Ontologien dienen bereits in breitem Maße zur Darstellung der Realität. Für die Sichtenskizzen und die ontologischen Einheiten werden entsprechende Integritätsbedingungen angegeben. Die Verfeinerung des Lastenheftes findet durch Spezialisierung der Typen, Dekomposition, strukturelle Erweiterung, semantische Einschränkung, Separation von Aspekten und durch Instantiierung statt. Zusätzlich werden weitere Typen eingeführt.

Die Sichtenskizze enthält die Spezifikation der Darstellung der wichtigen Typen und eine grobe Vorstellung über die Art der Benutzung der Sichten. Es wird wiederum der Zusammenhang zur Darstellung der Strukturierung und der Funktionalität im Pflichtenheft hergestellt. Alle Ereignisse des Handlungsrahmens werden durch entsprechende Teile der Sichtenskizze unterstützt.

Auf der Grundlage des Zusammenhangs zu verschiedenen Elementen der Story werden auch Zusammenhänge zwischen den einzelnen Sichten erkannt. Wir spezifizieren die Zusammenhänge in einem *Integrationsschema* der Sichten. Die Kohäsion zwischen den Sichten ist ein wichtiger Hinweis für eine spätere Sichtenintegration. Damit wird eine Bereinigung von Integrationskonflikten später vereinfacht und algorithmisch beherrschbar.

Aktionssichten-Suite: Eine *Suite* besteht aus einer Menge von Elementen, einem Integrations- bzw. Zusammenhangsschema zur Pflege des Zusammenhanges und Obligationen. Die Aktionssichten stellen die Strukturierung der Daten in einer Form dar, wie sie der Benutzer sehen wird. Dazu werden die *Kerntypen* dargestellt. Aus den Kerntypen können wir alle Sichtenskelette zusammenstellen. Damit werden durch die Sichtenskelette alle Typen repräsentiert, die für den Anwender eine Bedeutung haben. Die Typen stellen eine Verfeinerung der Typen des Pflichtenhefts dar oder sind neu eingeführt. Die Aktionssichten-Suite besteht aus den Sichtenskeletten mit den Kerntypen und aus dem weiterentwickelten Integrationsschema.

Die Sichtenskelette werden in Übereinstimmung mit dem Storyboard und dem Anwendungsschema entwickelt. Eine Spezifikation der einzelnen Sicht kann eine vollständige Erfassung aller wesentlichen Typen mit einschließen, so daß dieser Entwurfsschritt analog zur Spezifikation des

Anwendungsschemas geführt werden kann. Falls ein Anwendungsschema vorliegt, dann sollte jede Sicht auch als Anfrage über dem Anwendungsschema formuliert werden.

Durch die Informationen aus dem Storyboard und den Zusammenhangs der Sichten können Obligationen für den Entwurfsprozeß abgeleitet werden. Eine Bereinigung von Integrationskonflikten kann auf der Grundlage des Sichtenintegrationsschemas erfolgen. Deshalb wird dieses Schema weiter parallel verfeinert.

Sichten-Suite: Die Sichten-Suite stellt auf der konzeptionellen Schicht eine Menge integrierter Sichtenschemata dar, die auch durch entsprechende Strukturen des ER-Schemas und durch Anfragemengen unterstützt wird. Die einzelnen Sichten werden nun im Detail entworfen. Für jeden Typ einer Sicht wird angegeben, ob dieser Typ aus der Sicht der Datenbank ein Inputtyp, ein Outputtyp oder ein Modifikationstyp ist.

Auf der konzeptionellen Schicht werden die Typen für die Strukturierung durch ein detailliertes HERM-Diagramm angegeben. Diese Typen stellen eine Verfeinerung der Typen des Anwendungsschemas dar. Die Verfeinerungsbeziehung wird direkt zur Erzeugung der Sichten-Suite genutzt. Der Entwurf der Sichten kann nach den Entwurfsmethodiken des konzeptionellen Schemas angestrebt werden.

Bei Bereinigung von Integrationskonflikten kann nun auch eine Sichtenintegration angestrebt werden. Da uns das Integrationsschema bekannt ist und dieses fortgeschrieben wurde, kann eine Integration durch Generalisierung, durch Verschmelzung und Kombination oder im Extremfall durch Kooperation der Sichten angestrebt werden.

Die Sichten werden am Ende des konzeptionellen Entwurfes vollständig in das konzeptionelle ER-Schema und das Drehbuch eingebettet sein.

Zusätzlich zu den entwickelten Sichten werden die Sicherungssichten und die DBMS-Interaktionssichten entwickelt.

Logische Sichten-Suite: Die Sichten-Suite wird je nach gewählten Transformationsmodus für die Abbildung des ER-Schemas auf das logische Schema im Anschluß an die Transformation des ER-Schemas auf Sichtenkonzepte des DBMS bzw. der Plattform abgebildet. Dazu werden entsprechende Operationen, Programme oder Module der Datenbank-Maschine verwendet.

Die Sichtenkonzepte werden je nach Funktionalität des DBMS als externe Sichten oder materialisierte Sichten in die Beschreibung der Struktur der Datenbank eingebettet. Aus den konzeptionellen Sichten kann durch Transformation die jeweilige logische bzw. bei einer Materialisierung die physische Sicht erzeugt werden.

Relationale DBMS unterstützen oft nur typ-basierte Sichten. In diesem Fall wird für jeden Typ einer Sicht eine Sichtentypanfrage angegeben. Der Zusammenhang der Sichten wird mit einer integrierten Sichtenanfragemenge in der logischen Sichten-Suite gewährleistet. Werden semi-strukturierte Datenbank-Maschinen verwendet, dann kann auch eine Sicht z.B. durch eine DTD angegeben werden. Der Zusammenhang innerhalb einer Sicht wird dann durch XML-Dokumente direkt dargestellt. Unterschiedliche Sichtweisen auf ein XML-Dokument können durch entsprechende XSL-Regeln unterstützt werden.

Wir können die einzelnen Schritte wie in Bild 33 darstellen.

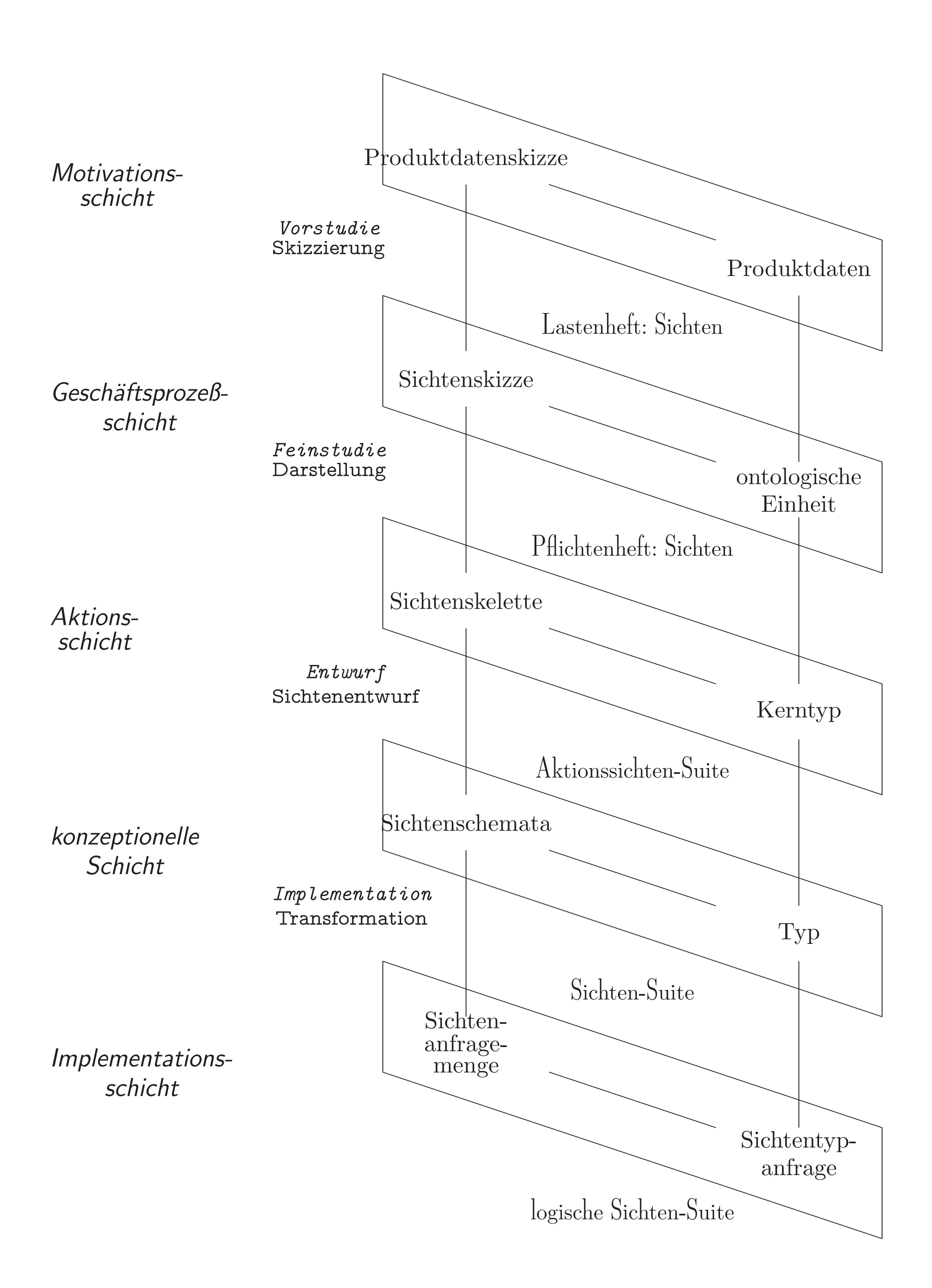


Bild 33: Die Arbeitsprodukte im Abstraktionsschichtenmodell für die Sichtenentwicklung

Sichten und Content-Typen

Sichten werden klassischerweise durch Anfragen über Datenbank-Schemata definiert. In diesem Fall benutzen wir als Rahmen:

```
select    Projektionsausdruck
  from        Datenbank-Struktur
  where       Auswahl-Bedingung
group by    Zusammenfassungsausdruck zu Gruppe
  having    Auswahl unter den Gruppen
order by  Lexikographische Ordnung unter Teilstruktur
```

Dieser Rahmen erlaubt die Definition einfacher Sichten, die auf einem Typ definiert sind. Damit ist jedoch eine konzeptionelle Darstellung zusammengehörender Objekte für die Ausgabe nicht möglich. Wir nutzen diesen Rahmen für die Definition der logischen Sichten.

Im allgemeinen benötigen wir jedoch in Anwendungen komplexere Unterstützung:

Spezifkation einer Sichten-Suite: Zur Begleitung der unterschiedlichen Arbeitsschritte sind auch unterschiedliche zusammenhängende Sichten zu definieren.

Spezifikation einer Funktionalität für die Sichten-Suite: Es sollte möglich sein, eine Anwendung soweit wie möglich durch entsprechende Funktionen und Prozesse zu unterstützen. Dazu benötigt ein Benutzer eine Reihe von Funktionen.

Spezifikation der Anpassung an den aktuellen oder potentiellen Benutzer: Jeder Akteur oder jeder aktuelle Benutzer sollte ggf. auch mit seiner Oberfläche arbeiten können, ggf. seine Daten auch für sich selbst modifizieren können und auch durch eine explizite Beschreibung der Präsentationsart eine Anpassung vornehmen können.

Die aktuell verfügbare Datenbank-Technologie unterstützt diese Forderungen bereits in breitem Maße, wenn Sichten-Suite-Modifikation über stored procedures abefangen wird. Sichten-Suiten können auch durch (logische) Sichtenanfragemengen unterstützt werden. Die Funktionen sind mit einem allgemeinen Funktionsrahmen allgemein darstellbar und dann an die konkrete Sichten-Suite anpaßbar. Die XML-Technologie eignet sich besonders für unterschiedliche Arten des "Ausspielens". Außerdem kann einem Benutzer auch ein Sitzungsobjekt zugeordnet werden, so daß entsprechende Einstellungen automatisch weitergeführt werden können. Sitzungsobjekte können direkt realisiert werden oder mit einem Verpackungsumschlag in das verpackte Content-Objekt integriert werden. Funktionen wie Markierungsfunktionen sind durch Sichten, die über materialisierten Sichten entstehen, darstellbar. Deshalb ist keine Neuentwicklung notwendig, sondern nur ein Spezifikationsrahmen zur Verfügung zu stellen.

Dieser Rahmen kann zu folgendem Rahmen verallgemeinert werden:

```
generate Mapping : Vars → Ausgabestruktur
     from Datenbanktypen
     where Auswahlbedingung
     represent using Allgemeiner Präsentationsstil
          & Abstraktion (Granularität, Masseinheit, Präzision)
          & Ordnungen der Präsentation
          & Hierarchische Darstellungen
          & Sichtweisen
          & Separation
     browsing definition Bedingung
               & Navigation
     functions Suchfunktionen
          & Exportfunktionen
          & Eingabefunktionen
          & Sitzungsverwaltung
          & Markierungsfunktionen
```

Damit können wir für die Sichten-Suite in einem Vierschritt-Verfahren die Spezifikation erstellen. Das Resultat dieses Spezifikationsprozesses nennen wir Content-Typ. Eine Instanz eines Content-Typs nennen wir Content-Objekt. Eine Content-Klasse enthält demzufolge Content-Objekte des gleichen Content-Typs. Die Spezifikation eines Content-Typs erfolgt durch schrittweise Erweiterung.

Wir unterscheiden drei Gesichtspunkte:

Content-Objekte werden den Akteuren zur Verfügung gestellt. Sie enthalten die Spezifikation der Strukturierung der dem Akteur zur Verfügung gestellten Daten und die Darstellung der Funktionalität.

Damit wird folgendes dargestellt:

Daten innerhalb von Content-Objekten sind in eine Reihe von Kategorien klassifizierbar:

Retrievaldaten, die aus einer Datenbank gewonnen werden und als Inputdaten für den ablaufenden Prozeß bzw. Dialogschritt dienen,

Inputdaten des Akteurs, die ggf. auch als Insert- oder Update-Daten in Dialogschritten fungieren,

Outputdaten, die in die Datenbank zurückgeschrieben werden,

Displaydaten, die als Output während des Dialoges dargestellt werden, und

Begleitdaten, die aus vorherigen Prozessen stammen und der Darstellung der Informationen während des Dialogschrittes dienen.

Bei Prozessen, mit denen ein Akteur Handlungen und Aktionen mit dem Informationssystem ausführen kann, unterscheiden wir:

Unterstützende Prozesse für die Aktionen und

Manipulationsanforderungen an das Informationssystem, die zur Veränderung der Daten führen können.

Wir fassen die Daten in Klassen zusammen. Ein Content-Typ spezifiziert eine solche Klasse und basiert auf einem Sichtenschema, das auch um die erforderliche Funktionalität angereichert wurde.

Container werden benutzt, um die Sichten den Akteuren bereitzustellen. Sie umfassen auch Parameter zur Beschreibung des Benutzungskontextes, so daß mit einer Auslieferung des Containers an den aktuellen Benutzer eine Adaption erfolgen kann.

Für die Beschreibung von Containern unterscheiden wir zwischen

allgemeiner Containerfunktionalität mit der Beschreibung allgemeiner Container-Funktionen zur Ent- und Beladung und

spezifischer Containerfunktionalität, die durch die Content-Objekte, mit denen ein Container beladen wurde, geprägt wird.

Ein Beispiel eines Content-Typen

Als Beispiel für ein Content-Objekt betrachten wir einen Archivierungstyp. Dieser Typ soll in unserem Hauptbeispiel benutzt werden, um aus der Datenbank zur Stundenplanung heraus eine Datenbank zur Ablage der relevanten Informationen zu gehaltenen Vorlesungen integriert mit entstehen zu lassen. Diese Datenbank dient der Verwaltung von Studentendaten, insbesondere für Informationen zu den erworbenen Scheinen. Die Funktionalität dieser Sicht ist stark eingeschränkt. Es können unterschiedliche Präsentationstil-Optionen zur Laufzeit gewählt werden.

Dieses Content-Objekt ist als Sicht über der Struktur in Bild 20 darstellbar. Die Archivsicht ist ein Ausschnitt der Daten. Die Daten, die nur für die Planung im laufenden oder kommenden Semester von Bedeutung sind, werden nicht archiviert.

Mithilfe der archivierten Daten können zu einem späteren Zeitpunkt die Daten zu Lehrveranstaltungen eingesehen werden, die stattfanden und in denen Studenten entsprechende Leistungen erreicht

haben. Lehrveranstaltungen, die stattfanden, in denen aber Studenten keine Abschlüsse erreichten, werden ebenfalls gespeichert. Sie sind jedoch für die Archivsicht nicht mehr von Interesse.
Die Archivsicht wird über dem Schema in Bild 20 als allgemeiner, parametrischer Ausdruck

Archiv(@SemesterBezeichnung)

mit obigem Rahmen spezifiziert. Sie wird instantiiert mit

Archiv("SS01/02") .

Der erste Teil der Sichtendefinition lautet somit:

$$\texttt{generate} \quad t_{Person} \mapsto Person\ ,\ t_{Kurs} \mapsto Kurs\ ,\ t_{gehalteneLV} \mapsto gehalteneLehrveranstaltung\ ,\ t_{Studiengang} \mapsto Studiengang\ ,\ t_{Typus} \mapsto Typus,\ t_{Professor} \mapsto Professor$$

$$\texttt{from} \quad t_{Kurs} := gehalteneLV\ [Kurs],$$
$$t_{Person} := gehalteneLV[geplanteLV[angeboteneLV[Verantwortlicher4LV{:}Person]]]),$$
$$t_{Studiengang} := ...,\ t_{Typus} := ...,,\ t_{Professor} := ...,$$
$$t_{gehalteneLV} := gehalteneLV[geplanteLV[\ angeboteneLV[\ Kurs, Studiengang, Dozent{:}Professor, Verantwortlicher4LV{:}Person], Typus]])$$

$$\texttt{where} \quad Bezeichnung = @SemesterBezeichnung\ ;$$

Sie ist mit einem Parameter *Semester* als materialisierte Read-Only-Sicht in Bild 34 dargestellt. Mit dieser Sicht ist eine Modifikation der Daten nicht mehr erlaubt. Sie kann nur als Anfragesicht verwendet werden.

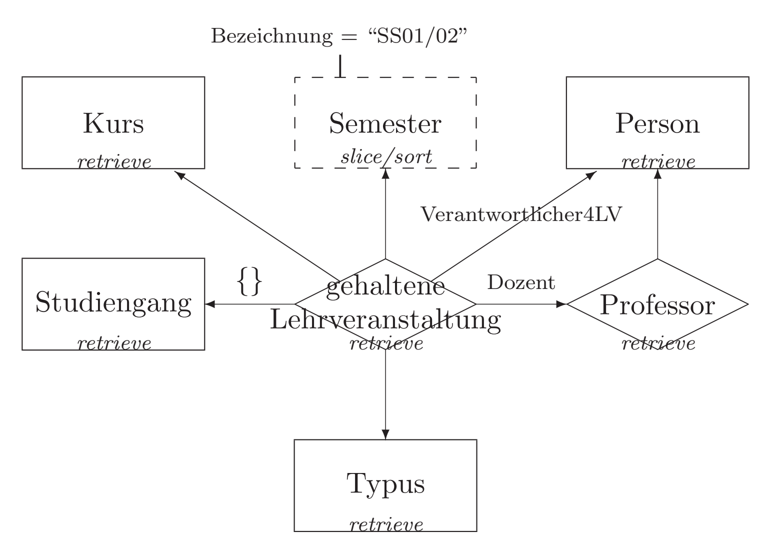


Bild 34: Content-Typen zur Archivsicht auf gehaltene Lehrveranstaltungen

Darstellung von Sichtenschemata

Wir erweitern die Darstellung von ER-Schemata wie bereits z.T. in Bild 34 verwendet:

Optionale Komponenten sind für Relationship-Typen von Sichten zugelassen. Sie werden mit einer gestrichelten Linie angegeben.

Versteckte Komponenten sind in einer Sicht in der Definition vorhanden, werden aber nicht angezeigt. Sie werden mit einem gestrichelten Typ mit dem Auswahlprädikat dargestellt.

Default-Werte werden für eine Sicht für die Generierung der Sicht benutzt. Sie können jedoch im Dialog durch andere Werte ersetzt werden. Es wird für einen Typ der Default-Wert mit der Identifikation des Typs angezeigt.

Wir merken an, daß sich mit einer Sichtendefinition auch die Integritätsbedingungen für die Typen einer Sicht ändern können.

Wir verwenden das Sichtenschema, um die Funktionalität der einzelnen Typen mit anzugeben. Damit wird ein schnellerer Überblick gegeben.

Erstellung der Sichten-Suite

Es werden die Sichten als konzeptionelle Sichten in ihrem Zusammenhang, mit einer Erweiterung um ggf. andere Datenbestände, sowie um andere Datentypen wie z.B. den Basis-Datentyp `URL`, `money` und `MAIL` dargestellt.

Ein Sichtenschema wird als ER-Schema dargestellt, in dem jedem Typ eine Anfrage über dem ER-Schema und den Typenerweiterungen zugeordnet wird.

Ein Sichtenschema kann auch materialisiert abgelegt werden. Dazu ist anzugeben, auf welche Art eine Modifikation in der Datenbank sich auf die Sicht auswirkt. Diese Materialisierung nutzt dann das folgende Schema:

```
extend Sichtenschema
     by MODIFIKATIONSMODUS
     store ABLAGE-SCHEMA
```

Wir werden im weiteren diesen Spezifikationsrahmen erweitern um eine Steuerbedingung

`accept on` ABSCHLUSSBEDINGUNG

mit der eine Kontrolle der Integrität dynamisch erfolgen kann. Eine derartige Kontrolle verbessert die Übersichtlichkeit, erfordert aber eine rigidere Behandlung der Konsistenz aller Integritätsbedingungen.

Für den Modifikationsmodus erstellen wir uns parametrische aktive Datenbank-Trigger. Diese parametrischen Trigger besitzen einen Namen, sind für Modifikationsoperationen über der Datenbank spezifiziert, können bei Gültigkeit einer Bedingung aktiviert werden und führen Aktionen zur Veränderung einer materialisierten Sicht aus.

Der Modifikationsmodus besteht aus einem Modifikationsschema und einem Zeitschema. Das Modifikationsschema kann durch entsprechende Triggeroperationen in der logischen Sichten-Suite unterlegt werden in der Form

```
on Modify on Datenbank-Schema-Typ
    if  Sichten des Sichtenschemas XY betroffen
    do  Modify XY
```

`Modify` steht für `Insert`, `Delete` bzw. `Update`.
Das Zeitschema diktiert, wann eine Modifikation der Sichten erfolgt. Default-Wert kann z.B. `Immediate` sein. Mitunter ist auch Aktionen `DeferUntilNoUserActive` sinnvoll.

Das Ablage-Schema kann sowohl eine einzelne URL bzw. URI als auch eine Menge zulassen, falls eine redundante Speicherung erforderlich ist.

Für die Archivsicht erhalten wir mit Darstellung durch den deontischen Operator F (Verboten):

```
extend Archivsicht
     by MODIFIKATION = { F Insert, F Delete, F Update }
     store ARCHIVSICHT
```

Weiterhin wird ein Sichtenschema durch die Angabe aufbereitet, ob die Objekte dieser Sicht nur Daten sind, die dem Benutzer zu Ansicht zur Verfügung (*Retrievaldaten*) stehen oder auch zur Modifikation (*Modifikationsdaten*) benutzbar sind.

Objekte einer Bearbeitung mit Sichten enthalten in der obigen Klassifikation demzufolge:

- *Input-Daten*, die dem Benutzer in den einzelnen Arbeitsschritten zur Verfügung gestellt werden,
- *Output-Daten*, mit denen der Benutzer auch Daten wieder in das System zurückschreiben oder einschreiben kann, und
- *Begleit-Daten*, die der Benutzer einsehen, aber nicht modifizieren kann.

Außerdem können Objekte einer Sicht sichtbar sein (*Displaydaten*) oder auch nicht dem Akteur sichtbar gemacht werden. Insbesondere wollen wir damit die direkte Modifikation der Objekte der Datenbank unterstützen, ohne dem Benutzer auch die Identifikation der Daten bekannt zu machen.

Damit ist das Sichtenschema um die Angabe

```
type  .... for modification
           for retrieval
      used for input
           for output
           for escort only
      displayed with subtype
```

erweitert.
Um die Identifizierbarkeit zu gewährleisten, verwenden wir dabei evt. auch Typen, die dem Benutzer nicht angegeben werden. Weiterhin können diese allgemeineren Typen auch für die Spezifikation der Funktionen verwendet werden.

Anreicherung der Sichtenschemata um Funktionen

Eine Funktion ist allgemein mit einem Definitionsrahmen der folgenden Form spezifiziert:

```
Signatur der Funktion: Name, Input-Parameter, Output-Parameter
     Basiert auf Sichtenschema
  Deklaration der Funktion
```

Dieser Definitionsrahmen kann für jede Art von Funktionsdefinition verwendet werden. In vereinfachter Form kann auch der folgende Definitionsrahmen verwendet werden:

`extend` Sichtenname
  `by functions` Kategorie der Funktionen
    `Name der Funktion` (Input-Parameter, Output-Parameter)
    Deklaration der Funktion

In diesem Fall wird die Erweiterung der Sichtendefinition hinzugefügt.

Wir benötigen in internet-basierten Anwendungen eine ganze Reihe unterschiedlicher Funktionen, die wir wie folgt klassifizieren können:

Durchmusterungsfunktionen erlauben die Erschließung von größeren Datenmengen ohne Verlust der Orientierung. Dazu gehören:

- Suchfunktionen, mit denen die Sichten und deren Objekte durchsucht werden können,
- Generalisierungs- und Spezialisierungsfunktionen (zooming-out, zooming-in), mit denen eine Menge von Objekten zu einer abstrakten Menge zusammengefaßt sowie diese Zusammenfassungen wieder aufgehoben werden können,
- Umordnungsfunktionen, mit denen eine Menge von Objekten auch in einer anderen Ordnung dargestellt werden kann,
- Navigationsfunktionen, (browsing, zapping, n-by-n, object-by-object) mit denen Objektmengen schrittweise, bündelweise, im Schnelldurchgang oder auch mit einem Browser durchmustert werden können,
- Kontexterschließungsfunktionen, mit denen die assoziierten Objekte zu einer Objektmenge erfaßt und dann mit den Objekten der Objektmenge verbunden werden können,
- Überblicksfunktionen, die anhand von Klassifikationskriterien die Erstellung einer 'Datenlandkarte' unterstützen, und
- Assoziationsfunktionen, mit denen Objekte aufgrund von Assoziationsbeziehungen schrittweise zu komplexeren Objekten umgeformt werden können,

In der Archivsicht können wir folgende Funktionen einführen:

```
extend Archivsicht
     by functions SUCHFUNKTIONEN
          Lehrveranstaltungsübersicht ((Von, Bis, Kurs.Name),
                                       (Verantwortliche, Semester.Bezeichnung) )
               stored procedure := ...
          Lehrveranstaltungen der Architektur ((), (Kurs.Name) )
               stored procedure := ...
     by functions NAVIGATIONSFUNKTIONEN
          Semesterübersicht ((Semester.Bezeichnung), )
               browse by Studiengang, Typus
     by functions ASSOZIATIONSFUNKTIONEN
          Vorlesungsprofil ((Professor.Name),(LV-Übersicht) )
               view defined as ...
```

Die Suchfunktionen sollen eine vereinfachte Suche unterstützen. Die Navigationsfunktionen werden für eine begleitende Navigation für die Oberflächen der Benutzer erstellt. Die Assoziationsfunktion erlaubt die Erstellung eines *Profils* mit einer neuen Sicht.

Bearbeitungsfunktionen ermöglichen die Bearbeitung von Daten aus der Datenbank, von Sichtendaten und von persönlichen Daten der Benutzer.

- **Datenbank-Modifikationsoperationen** erlauben dem Akteur, seine Daten in die Datenbank einzubringen, in der Datenbank Informationen zum Arbeitsverlauf vorzuhalten und Daten aus Sichten nach ihrer Bearbeitung durch den Benutzer in die Datenbank zurückzuspeichern.
- **Sichten-Objekt-Modifikationsoperationen** ermöglichen eine (temporäre) Veränderung der Daten in den Sichten. Diese Daten können materialisiert oder mit dem Erlöschen der Sicht auch gelöscht werden.
- **Bearbeitungsfunktionen für den eigenen Arbeitsplatz** unterstützen die Bearbeitung von Containern zu Sitzungen, die temporäre Haltung von Daten, das Einlagern, Modifizieren und Streichen von eigenen Daten.

Integrationsfunktionen erlauben dem Benutzer, aus dem Dialogverlauf heraus für sich Daten zu entnehmen bzw. einzubringen.
Exportfunktionen sollen eine ganze Reihe von Funktionen unterstützen:

- **Ausgabe in Druck-Dokumente:** Eine Druckausgabefunktion erlaubt die Ausgabe in vorgegebener Form z.B. als Formatting Object. Damit wird einem Benutzer nicht nur der Inhalt seiner Sitzung oder seiner Arbeitssichten bereitgestellt, sondern es werden auch die Daten des Content-Objektes für eine Ausgabe aufbereitet.
- **Integration in andere Dokumente:** Mitunter sind die Auswahl von Daten, die erarbeiteten Daten oder Sichten auf das Content-Objekt auch in andere Objekte integrierbar. In unserem Hauptbeispiel sollte z.B. die Übernahme von *Kursbeschreibungen* möglich sein.
- **Integration in den eigenen Arbeitsraum:** Es können Sichten auf das Content-Objekt und die Sitzung in den eigenen Arbeitsraum des Benutzers eingelagert werden.
- **Weitergabe von bearbeiteten Objekten an andere Akteure:** Eine Weitergabefunktion von Arbeitsresultaten kann in analoger Form wie die Druckfunktion realisiert sein.

In analoger Form können auch **Importfunktionen** bereitgestellt werden. Sie unterstützen den Akteur in den entsprechenden Dialogschritten und basieren auf folgenden Funktionen:

- **Übernahme von Objekten in die Datenbank:** Eine Eingabe sollte nicht nur textuell erfolgen, sondern durch Funktionen zur Übernahme von Dokumenten oder Mengen von Objekten unterstützt werden. Dazu werden Techniken der Sichten-Kooperation genutzt.

Integration in das Content-Objekt: Das Content-Objekt kann Parameter haben, die durch eine Eingabe von Daten oder Objekten instantiiert werden können. Damit ist auch eine Erweiterung des aktuellen Content-Objektes der aktuellen Sitzung möglich.

Integration in den eigenen Arbeitsraum: Es können in vorher vereinbarten Formaten durch entsprechende Importfunktionen auch entsprechende Content-Objekte des Benutzers benutzt und in den Arbeitsraum eingebracht werden.

Integration in die Arbeitssichten: Den Parametern aktueller Arbeitssichten können auch entsprechende Content-Objekte zugewiesen werden, so daß mit einer Importfunktion auch das aktuelle Content-Objekt verändert wird.

Sowohl Import- als auch Exportfunktionen können auf der sogenannten Wrapper-Technologie aufsetzen. Wir verwenden zur einfacheren Integration die unten dargestellten Mechanismen der Sichtenkooperation.

In unserer Anwendung kann z.B. die Archivsicht um Funktionen zum Druck wie folgt erweitert werden:

`extend` *Archivsicht*
  `by functions` EXPORTFUNKTIONEN
    `ProfilübersichtPDF` *((Professor.Name), (Dokument))*
     ... `Vorlesungsprofil` *((Professor.Name),(LV-Übersicht))* ...

Markierungsfunktionen erlauben dem Benutzer mit den dargestellten Daten so umzugehen wie mit Daten auf seinem Schreibtisch. Es kann dem Akteur eine sehr breite Palette zur Verfügung gestellt werden. Oft verwendet werden Funktionen wie

Kopierfunktionen zum Kopieren von Daten in den eigenen Arbeitsraum,

Färbungsfunktionen zum Markieren von Daten mit unterschiedlichen Beschriftungen wie z.B. Farben,

Beschriftungsfunktionen zur Annotation, zum Einbringen von Kommentaren und zum Anbringen von Variationen.

Markierungsfunktionen können durch einen benutzereigenen Container unterstützt werden. Container werden auf Seite 96 eingeführt.

Funktionen zur Sitzungsverwaltung erlauben einem Benutzer auch die Wiederaufnahme der Arbeit an der entsprechenden Stelle. Jedem Benutzer wird in seinem Arbeitsraum auch ein Sitzung-Container zur Verfügung gestellt. In diesem Container werden die Sitzungen mitprotokolliert. Damit ist dann auch eine Weiterführung eines bereits partiell durchlaufenen Workflows möglich. Funktionen der Sitzungsverwaltung sind insbesondere:

Funktionen zum Öffnen, Protokollieren und Schließen von Sitzungen, mit denen ein Arbeitsstand gespeichert werden kann, die Erhaltung persönlicher Daten gewährleistet wird, mit denen Nebensitzungen und Gruppensitzungen unterstützt werden,

Absicherungsfunktionen zur Absicherung der Sitzungsinformation und der Workspace-Information vor unberechtigtem Zugriff oder unberechtigter Modifikation,

Weitergabefunktionen, mit denen Sitzungsinformationen an andere Benutzer oder Funktionen weitergegeben bzw. andere Benutzer kontaktiert werden können, und

Löschfunktionen, mit denen ältere Sitzungen gelöscht werden können.

Eine einfache Form der Sitzungsverwaltung stellen Cookies dar.

Neben diesen Funktionen können auch Funktionen für *Gruppensitzungen* zur Verfügung gestellt werden. Diese Funktionen unterstützen eine effiziente Arbeit von Gruppen wie z.B. Gremien, Versammlungen, Veranstaltungen, etc. durch eine Reihe von Funktionen wie:

Funktionen zur Darstellung der Arbeit der Gruppen mit variabler Sichtbarkeit der Tagesordnungen, Dokumente und Nachrichten je nach Freigabe,

Funktionen zur Veröffentlichung von Materialien und Dokumenten mit unterschiedlicher Sichtbarkeit und unterschiedlichem Recht auf Einsicht,

Funktionen zur Unterstützung der Zusammenarbeit von Mitgliedern der Gruppe untereinander bzw. mit den interessierten Akteuren und

Funktionen zur Archivierung der Materialien mit unterschiedlicher Einsicht in die Dokumente je nach Rechten und je nach Freigabestatus.

Diese Funktionsmenge ist bereits in einer Reihe von Anwendungen in generischer Form entwickelt worden. So kann z.B. Das CPAN-Verzeichnis zu Perl-Anwendungen auch zur schnellen Entwicklung der erforderlichen Funktionalität für Sichten-Suiten herangezogen werden.

Parametrisierte Anpassung an den Akteur

Um Benutzern in ihren Rollen entgegenzukommen, sollen Content-Objekte in gewissem Maße an folgende Dinge adaptierbar sein:

- an den Benutzer, insbesondere an das Benutzerprofil, die Sprache, seine Kenntnisse und Fertigkeiten, seine Präferenzen,
- an das Benutzungsportfolio, d.h. die Arbeitsaufgaben des Benutzers,
- an die Arbeitsumgebung des Benutzers, insbesondere die technische Infrastruktur wie Hard- und Software der Arbeitsplatzrechner, die Kommunikationsinfrastruktur und
- die Benutzungsgeschichte.

Eine solche Anpassung ist nicht im allgemeinen Maße möglich. Ein Sichtenschema ist jedoch parametrisierbar und im Rahmen dieser Parametrisierung an den konkreten Kontext adaptierbar.

Dazu erweitern wir die Spezifikation der Content-Typen:

Anwendbare Abstraktionen innerhalb des Sichtenschemas: Zur Unterstützung der Suche und der Navigation innerhalb des Content-Objektes kann man das Wissen zu den verwendeten Datentypen einbringen. Jeder Basis-Datentyp kann auch in vergröberter Form dargestellt werden. Diese Vergröberung vererbt sich über das Konstruktionsschema bis hin zum Sichtenschema. Damit sind wir in der Lage,

- die Granularität,
- die verwendeten Maßeinheiten und
- die Präzision der Darstellung

anzupassen. Diese Anpassung wird in Spreadsheet-Zugängen bereits breit praktiziert und ist relativ einfach mit dem Content-Typ verbindbar.

Präsentationsstil: Der Datentyp des Sichtenschemas ist durch die verwendeten Datentypen gegeben. Wir können damit für einen allgemeinen Datentyp eine Menge von Präsentationsformaten entwickeln und mit dem Content-Objekt verknüpfen.

Allgemeiner Repräsentationsstil: Im Rahmen der Entwicklungen zu Benutzungsschnittstellen sind allgemeine *Gestaltungsraster* für die Präsentation entwickelt worden. Dazu gehört das Screen-Layout, die Typographie, die Integration von Metaphern in die Gestaltung nach allgemeinen Prinzipien:

- Das Prinzip der *visuellen Wahrnehmung* orientiert sich an Ordnungsbeziehungen innerhalb des Content-Typen, an Wirkungen der Darstellung und Hilfsmitteln zur Visualisierung,

- Das Prinzip der *visuellen Kommunikation* orientiert auf eine klare, konsistente Struktur mit einer minimalen Menge von Hilfsmitteln unter Berücksichtigung der Aufnahmefähigkeit des Akteurs.

Es werden dabei die Gestaltungsgesetze das Bildschirms und der visuellen Gestaltung auf die Darstellung der Content-Typen erweitert.

Der allgemeine Repräsentationsstil wird durch *style sheets* unterstützt. Darin werden nicht nur die Typographie und Farbkodierung festgelegt, sondern auch die genutzten Metaphern und Darstellungselemente. Diese können parametrisiert werden. Damit kann zur Laufzeit eine Adaption an den Benutzer erfolgen.

Inhaltsbasierter Repräsentationsstil: Durch die Konstruktion des Sichtenschemas können wir auch die Sichtendefinition und die Funktionalität des DBMS direkt nutzen, um unterschiedliche Darstellungsformen des gleichen Content-Objektes zu ermöglichen. Wir unterscheiden eine Reihe von Zugängen:

Zuordnung einer Menge von Sichten zum Content-Typ: Jedes Objekt kann auf unterschiedliche Art und Weise betrachtet werden. Die unterschiedlichen Gesichtspunkte auf das gleiche Objekt erlauben auch einen schnelleren und situationsbezogenen Überblick über das gleiche Content-Objekt. Das gleiche Objekt wird aus unterschiedlichen Sichtweisen dargestellt. Diese Sichtweisen werden durch Sichten, d.h. Ausdrücke der HERM-Algebra dargestellt.

Dieser Zugang wird in XML-Ausspielsystemen bereits praktiziert durch die Angabe von XSL-Regeln, die dann ein variables Darstellen des gleichen XML-Dokumentes erlauben.

Hierarchische Strukturierung als Darstellungshilfsmittel: Eine besondere Präsentationsform ist die hierarchische Darstellungsform. Wir können hierarchische Darstellungsformen einführen

- durch Verallgemeinerung von Zugängen, die für OLAP-Systeme entwickelt worden,
- durch Nutzung der Hierarchien, die bereits mit einer Assoziierung, wie z.B. Verweisen gegeben ist und
- durch Auflösung geschachtelter Strukturen.

Die Auflösung geschachtelter Strukturen ist bereits für die HERM-Algebra eingeführt worden.

Einführung von parametrischen Ordnungen: Bereits für die Durchmusterung und die Suche in größeren Content-Objekten ist eine Unterstützung durch entsprechende Funktionen entwickelt worden. Wir können diese Funktionen nutzen, um eine *Ordnungserweiterung* des Content-Typen vorzunehmen:

- Es werden die Ordnungsrelationen $ord_{\leq}$, die für Listen und Mengen bekannt sind, genutzt.
- Es werden Mengen durch Listen ersetzt und damit einfach sequentiell durchmusterbar.

Die Ordnungsrelationen sind von unterschiedlicher *Gewichtung*. Einige Eigenschaften charakterisieren ein Objekt stärker als andere. Deshalb können wir die Gewichtung auch für die Ordnung der Eigenschaften nutzen.

Das *Ordnungsschema* erlaubt eine Parametrisierung. Diese Parameter können zur Laufzeit durch entsprechende Ordnungen ersetzt werden. Dabei können auch bestimmte Ordnungen per Default zur Anwendung kommen. In unserer Vorlesungsanwendung können z.B. Lehrveranstaltungen nach *Vorlesungssemestern*, *Studiengängen* und *Studienabschniten* in dieser Reihenfolge geordnet werden.

Angabe von Dekompositionen bzw. Separationen des Content-Typen: Wir können den inneren Zusammenhang eines Content-Typen, der durch sein Sichtenschema gegeben ist, direkt verwenden. Ein Sichtenschema erlaubt eine Reihe von Sichtweisen auf die Daten. Diese Sichtweisen können als Sichten dem Sichtenschema zugeordnet werden. Welche Sichtweise auf das Content-Objekt durch den Benutzer gewählt wird, kann dann sogar

zur Laufzeit entschieden werden. Da nicht alle Typen des Sichtenschemas in der gleichen Form miteinander assoziiert sind, kann die *Stärke der Assoziierung* direkt mit den Typen verbunden werden.
Wir nutzen dafür eine *Adhäsionsmatrix*, die zwischen den Typen des Sichtenschemas definiert ist.

- Es sei $types(\mathfrak{S})$ die Menge aller Typen des Sichtenschemas $\mathfrak{S}$. Eine Adhäsionsmatrix $\mathcal{AM}$ ordnet jedem Paar von Typen $T, T' \in \mathfrak{S}$ eine natürliche Zahl oder 0 zur Darstellung des Abstandes zwischen den Typen in $\mathfrak{S}$ zu.
  Die Adhäsion ist umso niedriger, um so enger die Typen zusammengehören. Wir nehmen an, daß $\mathcal{AM}(T, T) = 0$ für jeden Typen T von $types(\mathfrak{S})$ gilt.
- Die Zuordnung muß nicht vollständig für Teiltypen eines Typen T angegeben sein. Ist $\mathcal{AM}(T_1, T_2)$ nicht definiert, dann nehmen wir als Adhäsion den Abstand in der Typendefinition der Typen des Schemas an. Ist ein Schema nicht zusammenhängend und ist keine Adhäsion unter den Elementen der nicht zusammenhängenden Teilschemata definiert, dann nehmen wir $\mathcal{AM}(T_1, T_2) = \infty$ an.
- Eine Adhäsionsmatrix ist konservativ, falls $\mathcal{AM}(T_1, T_2) \leq \mathcal{M}(T_1', T_2')$ für Typen T_1, T_2 von $types(\mathfrak{S})$ und Teiltypen T_1', T_2' von jeweils T_1, T_2.
- Eine Adhäsionsmatrix muß nicht symmetrisch sein. Teiltypen T_1, T_2 eines Typen T können dem Typen T unterschiedlich nahe stehen.

Durch eine Adhäsionsmatrix können wir für jeden Typen T von $types(\mathfrak{S})$ *Schalen* definieren durch

$$Shell(T, types(\mathfrak{S}), i) \;=\; \{\, T' \in types(\mathfrak{S}) \mid \mathcal{AM}(T, T') \leq i \,\}$$

Die Schalen erlauben eine automatische Separation, insbesondere im Falle eines nicht ausreichenden Darstellungsraumes auf dem Bildschirm. Mit der Adhäsionsmatrix wird dargestellt, welche Typen und Teiltypen gemeinsam auf dem Bildschirm erscheinen müssen und welche nicht unbedingt im Zusammenhang mit einem Typ dargestellt werden müssen.
Wir können die Schalen und deren Beziehungen als *Hypergraphen* wie in Bild 35 darstellen. Ein Hypergraph besteht aus Knoten V und Hyperkanten $H \subseteq 2^V$. In unserem Modell sind die Hyperkanten hierarchisch. Es existiert eine lexikographische Nummerierung $E_{1,i_1,\ldots,i_k}$ der Kanten in H, so daß $E_{\bar{i}} \subseteq E_{\bar{j}}$ genau dann gilt, wenn $\bar{i}$ der Beginn von $\bar{j}$ ist. Die Wurzel ist der Knoten E_1.
Eine andere Darstellung kann auch analog zu Bild 21 mit dem erweiterten ER-Modell angegeben werden, indem die Rauten als Relationship-Typen-Folge an der jeweils darunterliegenden Schale angehängt werden. In diesem Fall wird ein Stern-Schema erzeugt. Meist wird jedoch eine vollständige hierarchische Strukturierung nicht möglich sein. Dann erhalten wir ein Schneeflocken-Schema.
Ein Beispiel einer Adhäsionsmatrix für das Schema in Bild 34 ist mit folgender Matrix gegeben:

Archivsicht	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
T_1 = *Verantwortlicher*	0	2	0	4	4	2	5	11
T_2 = *Professor*	1	0	1	4	6	4	4	7
T_3 = *gehaltene Lehrveranstaltung*	2	1	0	2	4	1	3	5
T_4 = *Semester*	5	4	2	0	1	3	6	9
T_5 = *Bezeichnung*	6	6	4	1	0	4	7	10
T_6 = *Kurs*	4	3	0	3	3	0	2	5
T_7 = *Studiengang*	5	3	2	5	7	2	0	11
T_8 = *Typus*	6	2	1	7	9	1	3	0

Eine Adhäsionsmatrix kann auch per Default besetzt werden. Wir verwenden dazu den Abstand in der Typen-Definition. Die Beispielmatrix erlaubt z.B. eine Separation der Typen in Bild 34 für den Typ *Kurs* in die Schalen

Kurs, gehaltene Lehrveranstaltung

Kurs, gehaltene Lehrveranstaltung, Studiengang

Kurs, gehaltene Lehrveranstaltung, Studiengang, Professor

Kurs, gehaltene Lehrveranstaltung, Studiengang, Professor, Semester, Bezeichnung, Verantwortlicher und

die gesamte Sicht.

Die Schalen können auch noch gegenseitig durch die Adhäsion der hinzukommenden Typen der nächsten Schale separiert werden. Dadurch können wir sogar eine hierarchische Charakterisierung vornehmen. In den seltensten Fällen wird jedoch eine solche Detailliertheit benötigt. Im Beispiel in Bild 35 kann die vierte Schale z.B. in die *Personenangaben*, die Angabe der *Verantwortlichkeit* und die *Semesterangaben* separiert werden.

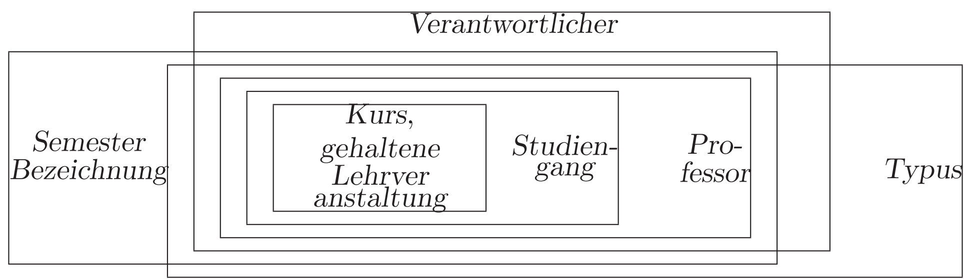


Bild 35: Hierarchische Schalen des Typen *Kurs* in der *Archivsicht*

Abstrakte und Verpackungsumschläge von Content-Objekten

Content-Objekte sind Objekte eines Content-Typs, die an den Akteur ausgeliefert werden und ihm zur Verfügung stehen. Ein Content-Objekt kann relativ groß werden. Deshalb kann ein Content-Objekt mit einer Beschreibung versehen werden, die über den Inhalt Auskunft gibt. Diese Beschreibung wird mit einer Extraktionsfunktion gewonnen.

Abstrakte dienen als verallgemeinerte Indizes und erlauben eine Vorausschau auf das Content-Objekt. Der Name des Content-Objektes wird um den Abstrakt erweitert. Abstrakte umfassen:

die Titel-Information nach einem Benennungsschema, mit einer Kurz-Identifikation,

eine Kurzbeschreibung des Inhaltes des Content-Objektes,

die Zusammenfassung des Inhaltes des Content-Objektes, die durch Anwendung entsprechender Extraktionsfunktionen des Content-Typen aus dem Content-Objekt gewonnen werden können,

allgemeine Beschreibungen des Inhaltes und der Strukturierung der Content-Objekte, einschließlich der Variablen,

weitere Informationen z.B. zu den Autoren und zu Klassifikation für das Content-Objekte,

Angaben zur Funktionalität des Content-Objektes, d.h.

- zu Durchmusterungsfunktionen,
- zu Integrationsfunktionen,
- zu Markierungsfunktionen und
- zu Funktionen zur Sitzungsverwaltung,

Prozeduren und Programmme zur Anpassung des Content-Objektes an den aktuellen Benutzer, d.h.

- zur Anwendung von Abstraktionen,
- zur Anpassung des allgemeinen Repräsentationsstils und

  - zur Anpassung an den inhaltsbasierten Repräsentationsstil.

Ein Content-Objekt wird

- ggf. von einem Akteur mehrfach benutzt,
- ggf. von einem Akteur mit entsprechenden Anmerkungen unter Benutzung der Markierungsfunktionen versehen und
- erfordert eine Aufzeichnung der Benutzungsgeschichte.

Diese Aufzeichnung wird mit einem Anhänger dem Content-Objekt zugeordnet. Verpackungsumschläge (Kuvert) (envelops, docket) dienen als Anhänger. Sie enthalten:

1. eine allgemeine Inhalts-Information, in der
   - die Sourcen, der Provider, die Autoren und die Benutzungsinformation mitgeführt werden,
   - der Inhalt und die unterstützten Aufgaben, die Eignung und die Art der Erzeugung dargestellt werden und
   - die Qualitätsbewertungen für das Content-Objekt angegeben werden,
2. eine Anwendungsanleitung für das Content-Objekt, die auch Anmerkungen zu folgenden Dingen umfaßt:
   - Vertrauenswürdigkeit, dem Umfang der bereitgestellten Information, der Benutzungsrechte, Sicherheitskriterien und den Geschäftsbedingungen,
   - assoziierten Content-Objekten für unterschiedliche Benutzergruppen und
   - Annotationen, Anmerkungen zu Zugriffsmodellen, spezifischen Annotationen, zum Ressourcentypen und -formaten, sowie zur verwendeten Sprache,
3. die Benutzungsgeschichte des Content-Objektes, die mit Parametern erfaßt und angepaßt werden kann, die schrittweise zu einer Erweiterung des Umschlags führen,
4. allgemeine Zeitinformation, insbesondere
   - zu Versionen, Ausgaben und Benutzungsprofilen,
   - zu Erneuerungsstrategien, anwendbaren Verbindungsprofilen zur Erneuerung und die Art der Verbindung und
   - Signaturen, Beglaubigungshinweisen und Angaben zur wiederholten Benutzung.

Wir fügen diesen Verpackungsinformationen dem Content-Objekt hinzu, indem durch Variable-Werte-Paare eine erweiterbare Attribut-Information mitgeführt wird.

Container für die Auslieferung von Content-Objekten

Content-Objekte sollen dem Benutzer zur Verfügung stehen. Dabei wollen wir eine möglichst große Unabhängigkeit von der aktuellen Web-Technologie erreichen. Eine Auslieferung von Content-Objekten kann sowohl über der Internet als auch das Extranet oder Intranet erfolgen. Weiterhin kann ein Benutzer die Daten mit einem komfortablen System, wie z.B. einem Browser, einem weniger komfortablen System, wie z.B. einen text-basierten Browser, einem eingeschränktem Medium, wie z.B. einem Wap-Handy oder auch mit einem interaktionsbeschränkten Medium, wie z.B. Tele-Text, entgegennehmen und bearbeiten können. Deshalb muß ein Auslieferungsmedium eine hohe Allgemeinheit und eine sehr hohe Anpaßbarkeit besitzen. Wir führen dazu den Begriff des Containers ein. Ein Container soll beladen, an den Benutzer versandt und von ihm benutzt werden können. Durch die enthaltenen Content-Objekte wird einem Benutzer die erforderliche Datenmengen und Funktionalität bereitgestellt.

Aufgrund dieser Anforderungen bedienen wir uns der Zugänge von Skriptsprachen. Dadurch kann auch eine Realisierung von Containern mit den Mitteln von Skriptsprachen erfolgen.

Ein Container wird durch eine abstrakte Zustandsmaschine beschrieben:

einem Namen $\mathfrak{C}$ zur Bezeichnung des Containers,

Zustandsräumen (Input-Raum, Content-Raum, Output-Raum) zur Aufnahme von Content-Objekten, die wir dem Benutzer zur Verfügung stellen wollen. Wir unterscheiden dabei drei verschiedene Räume:

- Input-Raum $\mathcal{I}$: Zur Beladung der Container mit Inhalten wird ein Input-Raum zur Verfügung gestellt.
- Output-Raum $\mathcal{O}$: Aus dem Container wird auf Anforderung des Benutzers ein passendes Content-Objekt ausgewählt und ihm zur Verfügung gestellt.
- Content-Raum $\mathcal{M}$: In einem Container befinden sich verpackte Content-Objekte. Diese haben die folgende Struktur:
  - Das Content-Objekt stellt die Daten und die Funktionalität, wie in diesem Abschnitt dargestellt, zur Verfügung.
  - Abstrakte zu Content-Objekten sind zusammenfassende Beschreibungen des Inhaltes. Sie können auch leer sein.
  - Kuverts erlauben die Führung von Begleit- und Benutzungsinformation zu Content-Objekten.

Operationen $ops_{\mathfrak{C}}$ sollen die Verwaltung der drei Zustandsräume unterstützen. Deshalb unterscheiden wir:

- Auswertungsfunktionen zur Einlagerung von Content-Objekten in den Container,
- Operationen zum Verändern des Zustandes des Containers,
- Operationen zum Anfordern von Content-Objekten aus dem Container.

Beschränkungen $\Sigma_{\mathfrak{C}}$ zum Container selbst sollen insbesondere darstellen

- das Vergleichsvermögen des Containers auf der Grundlage von Vergleichsmustern,
- die Beladungskapazität eines Containers und
- die Entladungsbeschränkungen für den Benutzungskontext.

Die Räume des Containers realisieren einen Tupel-Raum. Jedes Element hat die Form

$$(\mathit{Variable}, \mathit{Wert}) \quad .$$

Die Räume enthalten Multimengen von Elementen, d.h.

$$\begin{aligned} \mathcal{I} &= \{\!|\ t\ |\!\} \\ \mathcal{M} &= \{\!|\ t\ |\!\} \\ \mathcal{O} &= \{\!|\ t\ |\!\} \end{aligned}$$

Eine Unterscheidung von Elementen erfolgt durch eine Mustererkennung der Variablen.
Sind Elemente mehrfach in einem Container enthalten, dann muß eine intelligente Mustererkennung eine Separation erlauben.
Variable sind Worte eines Alphabetes *Alph*.
Variable können auch die Kuverts und Abstrakte aufnehmen.
Werte sind Content-Objekte.
Ein Container ist *konsistent* beladen, falls seine Tupel-Variablen eindeutig sind. Wir fordern jedoch keine Konsistenz a priori.
Ein Container verfügt über eine Muster-Vergleichsfunktion $\approx_{\mathfrak{C}}$, mit der Elemente verglichen werden können. Der Mustervergleich hängt von den Mustern M ab, die ein Container vergleichen kann. Dieser Mustervergleich wird benutzt, um die Annahme von Content-Objekten zu verweigern oder auch dem Benutzer für seine Spezifikation ein passendes Content-Objekt auszugeben.

Ein Vergleich von Elementen eines Containers nutzt ein Muster m unter Einbeziehung eines der Muster des Containers, wobei dann der Durchschnitt der beiden Muster zur Erkennung genutzt werden kann, und wird gültig für Elemente, falls keine Ungleichheit erkannt werden kann.

$$(v,w) \approx_{\mathfrak{C},m} (v',w') = \begin{cases} \mathit{true} & \text{falls } \exists m' \in M : v, v' \preceq m' \sqcap m \ \wedge\ (\ w = w' \ \vee w = \perp\ \vee w' = \perp) \\ \mathit{false} & \text{andernfalls} \end{cases}$$

Mit diesem allgemeinen Vergleich kann ein Container sowohl alle Elemente als nicht unterscheidbar betrachten ($M = \emptyset$) als auch alle Elemente genau unterscheiden ($M = Alph^+$).

Wir können nun die Operationen des Containers als parallel ablaufende Operationen zur Zustandveränderung behandeln. Diese Funktionen basieren auf folgenden Elementaroperationen des Containers:

- $eval(t)$ ist eine Auswertungsfunktion des Containers mit folgenden Eigenschaften:
  - $eval(t)$ kann ggf. die Aufnahme von Content-Objekten blockieren. In diesem Fall ist das Resultat eine leere Multimenge.
  - Die Auswertung eines Content-Objektes kann auch zur Dekomposition dieses Content-Objektes führen, weil die Beladungskapazität des Containers für Einzelelemente ggf. beschränkt ist.
  - Die Auswertungsfunktion kann entsprechende Zeit erfordern. Mit einem Prädikat $success(eval(t))$ wird der Erfolg gemeldet.
- $inspect(\mathfrak{C}, m, t) \;=\; \{\!\mid t' \mid t \approx_{\mathfrak{C},m} t' \mid\!\}$
- $choose(M)$ wählt ein Element aus einer Multimenge aus.

Wir benötigen nur vier Zustandsveränderungsfunktionen zur Veränderung von $\mathcal{Z} = (\mathcal{I}, \mathcal{M}, \mathcal{O})$ mit Elementen $t \in Tupel_{\mathfrak{C}}$ und Mustern $Muster$:

Schnelle Beladung: Die Funktion $load : Z \times Tupel_{\mathfrak{C}} \rightarrow Z$ mit

$$load((\mathcal{I}, \mathcal{M}, \mathcal{O}), t) \;=\; (\mathcal{I}, \mathcal{M} \sqcup \{\!\mid eval(t) \mid\!\}, \mathcal{O})$$

erlaubt eine sofortige Beladung von Containern.

Langsame Beladung: Die Funktion $lazzyload : Z \times Tupel_{\mathfrak{C}} \rightarrow Z$ mit

$$lazzyload((\mathcal{I}, \mathcal{M}, \mathcal{O}), t) \;= success(eval(t)) \Rightarrow (\mathcal{I}, \mathcal{M} \sqcup \{\!\mid eval(t) \mid\!\}, \mathcal{O})$$

unterstützt eine verzögerte Beladung ohne auf die Beendigung der Berechnung von $eval$ zu warten.

Lesen im Containers: Die Funktion $read : Z \times Muster \times Tupel_{\mathfrak{C}} \rightarrow \mathcal{O}$ mit

$$read((\mathcal{I}, \mathcal{M}, \mathcal{O}), m, t) \;=\; choose(inspect((\mathcal{I}, \mathcal{M}, \mathcal{O}), m, t))$$

generiert ein Resultat auf die Anfrage t mit dem Muster m.

Lesen und Löschen im Container: Die Funktion $read : Z \times Muster \times Tupel_{\mathfrak{C}} \rightarrow \mathcal{O} \times \mathcal{M}$ mit

$$read((\mathcal{I}, \mathcal{M}, \mathcal{O}), m, t) \;=\; \texttt{let} x := choose(inspect((\mathcal{I}, \mathcal{M}, \mathcal{O}), m, t)) \;:\; (x, \mathcal{M} \setminus \{\!\mid x \mid\!\})$$

generiert ein Resultat auf die Anfrage t mit dem Muster m und löscht dieses Resultat aus dem Content-Raum des Containers.

Wir haben die Definition des Containers und seiner Operationen so allgemein gehalten, damit wir Container sowohl mit CORBA oder anderen Middleware-Systemen als auch mit JavaBeans oder auch direkt mit Perl, PHP bzw. anderen Skriptsprachen realisieren können.

Diese Definition des Containers wird auch bei der Entwicklung von benutzereigenen Arbeitsräumen verwendet.

Container können verfeinert werden

- durch Instantiierung oder Adaption der Parameter
  - Vergrößerung und Verkleinerung der Kapazität,
  - Hinzufügen von Integritätsbedingungen und
  - Verfeinerung folgender Operationen:
    - der Vergleichsfunktion bzw. der Mustermenge,
    - der Auswertungsfunktion $eval$,
    - der Inspektionsfunktion $inspect$ und
    - der Auswahlfunktionen,
- sowie durch Verbesserung der Darstellung von
  - Abstrakten als Zusammenfassungen des Inhaltes der Content-Objekte und
  - Erweiterung der Kuverts, die wir im folgenden betrachten.

Die Verfeinerung führt aufgrund des generischen Charakters der Funktionen zu einer Veränderung des Verhaltens der vier Hauptfunktionen, nicht aber zur Veränderung der Funktionen.

Der Content-Typ Benutzer-Arbeitsplatz

Ein Informationssystem soll einen Benutzer effizient und effektiv in seiner Arbeit unterstützen. Das Portfolio, hauptsächlich bestehend aus dem Aufgabenmodell und dem Rollen- und Rechtemodellen des Akteurs, und das Benutzerprofil werden zur Generierung des Playout und des Layout der Content-Typen herangezogen. Portfolio und Profile behandeln wir im Abschnitt 7 ausführlich.

Weiterhin muß eine Unterstützung für die Zusammenarbeit in Arbeitsgruppen erfolgen. Damit soll ein Content-Typ "Arbeitsplatz" auch die Zusammenarbeit in Arbeitsgruppen und die Publikation der Resultate der Zusammenarbeit gewährleistet werden. Wir unterscheiden aktive Content-Objekte, aktivierte Content-Objekte und passive Content-Objekte und entwickeln Kooperationsverträge zwischen den Objekten. Prozesse und Dialoge der Content-Objekte können sich auch gegenseitig *bedingen, blockieren, abweisen* und *starten*.

Wir unterscheiden verschiedene Arten von *Kopplungsmechanismen*, die auch im Kombination verwendet werden können.

- Bei einer *Kopplung im Story-Raum* werden die gleichen Daten interaktiv verwendet. Die Operationen sind durch Interaktion gekoppelt. Dazu existieren verschiedene Kopplungsmethoden: *interne Kopplung, globale Kopplung, externe Kopplung, Kontrollflußkopplung, Wanderdatenkopplung* und *Parameterkopplung*.

- Die *Container-Kopplung* erlaubt nur ein Zusammenspiel der Content-Objekte unterschiedlicher Container. Es können verschiedene Grade der Kopplung unterschieden werden: *versteckte Kopplung, verstreute Kopplung* und *spezifizierte Kopplung*.

- Die *Kopplung durch Kooperation der Content-Objekte* im Sinne der Sichtenkooperation folgt der hierarchischen Struktur der Typen des Schemas. Je nach Erzwingungsmechanismus unterscheiden wir *Änderungskopplung* (Signaturänderungskopplung bzw. Implementationsänderungskopplung), *Verfeinerungskopplung* (Signaturverfeinerungskopplung, Implementationsverfeinerungskopplung) und *Erweiterungskopplung*.

Durch die *Kohäsion* wird die Bindung zwischen den einzelnen kooperierenden Objekten beschrieben. Aufgrund der Modellierung existieren verschiedene Arten. Die *Funktions-Kohäsion* (zufällige Kohäsion, logische Kohäsion, temporale Kohäsion, prozedurale Kohäsion, Kommunikationskohäsion, sequentielle Kohäsion und funktionale Kohäsion) geht von einer Bindung durch Operationen aus. Die *Typ-Kohäsion* (zerlegbare Kohäsion, mehrschichtige Kohäsion, nicht delegierte Kohäsion und verborgene Kohäsion) bewertet die Bindung der Objekte innerhalb einer Klasse. Die *Vererbungskohäsion* folgt der Definition der Hierarchien unter den Typen und Klassen.

Im Rahmen der Forschungen zur *Gruppenarbeit* (CSCW Computer supported cooperative work) wurden Dialage nach unterschiedlichen Eigenschaften charakterisiert.

Charakterisierung nach Raum und Zeit: Je nach Ort und Zeit sind unterschiedliche Dialoge möglich:

	Gleicher Ort	Anderer Ort
Gleiche Zeitpunkte	Elektronische Besprechung Elektronisches Brett Gemeinsamer Bildschirm Brainstorming Zuhörerreaktion	Videokonferenz Konversationsunterstützung Kooperatives Design Gruppeneditoren
Verschiedene Zeitpunkte	Gemeinsam genutzte Dateien Designwerkzeuge	Strukturierter Arbeitsfluß Elektrononische Post Nachrichtenbrett

Charakterisierung nach Interaktionsart: Die wichtigste Arten sind die folgenden:

Durch den ***Sprechakt*** wird die Interaktionsform beschrieben.

Im ***illokutionären Akt*** wird die kommunikative Funktion der menschlichen Kommunikation nachgebildet (z.B. präpositionaler Akt). Typische Darstellungsformen sind Assertion, Direktive, Kommissive, Deklarative und Expressive.

Für den ***perlokutionären Akt*** wird die Wirkung auf den Zuhörer bewertet.

Die Konversation ist eine Kombination einer Reihe von Sprechakten. Wir unterscheiden dabei die

die ***Konversation zur Handlung*** (Aufforderung zu einer Handlung),

die ***Konversation zur Klärung*** (als Interaktion),

die ***Konversation zur Entscheidung über Möglichkeiten*** (über einen Handlungsverlauf) und

die ***Konversation zur Orientierung*** (zur klareren Darstellung der Umgebung).

Die ***Charakterisierung nach Aktivitäten*** dient der Einbettung des Dialoges in die Spezifikation der Funktionalität.

Ein Content-Typ Benutzer-Arbeitsplatz sollte die eine oder die andere Form unterstützen. Wir wählen dazu einen Ansatz, der sich relativ einfach realisieren läßt, sich gleichzeitig harmonisch mit den bisherigen Ansätzen verbindet und in Bild 36 skizziert ist:

Kern-Typen des Content-Typs Benutzer-Arbeitsplatz sind die Typen *Person*, *Arbeitsgruppe* und *Arbeitsplatz*. Diesen Kern-Typen werden unterschiedliche Typen auf der Grundlage folgender Annahmen zugeordnet:

Gruppierung von Personen in Akteure: Personen werden je nach ihrem Portfolio (Aufgaben, Stelle, Rolle, Umstände und Ziele) gruppiert. Diese Abstraktion wird durch die Einführung von Akteuren unterstützt.

Arbeitgruppen: Eine Zusammenarbeit findet in Arbeitsgruppen und zwischen Arbeitsgruppen statt. Diese Interaktionsformen werden unterschieden. Die Mitarbeit von Personen in einer Arbeitsgruppe und das Treffen von Arbeitsgruppen sind durch unterschiedliche Typen realisiert.

Zuordnung der Rechte zu Akteuren: Akteure erhalten Rechte z.B. zur Veröffentlichung der Resultate. Die Rechte an der Bearbeitung von Content-Objekten können analog erfaßt werden.

Portfolio: Personen werden bei der Erledigung von Aufgaben unterstützt. Jede Person erhält dazu ihr spezifisches Portfolio, das in die Zusammenarbeit der Arbeitsgruppen einfließt.

Organisationsmodell: Es wird ein einfaches *Organisationsmodell* benutzt, bei dem einer Person Rollen zugeordnet werden, die in der Firma üblich sind.

Content-Objekte und Container stehen den Benutzern zur Verfügung. Sie befinden sich zu unterschiedlichen Zeitpunkten auf unterschiedlichen Arbeitsplätzen.

Mit einem Content-Typ Arbeitsplatz können sowohl Arbeitsgruppen, als auch Benutzer auf einfache Art in ihren Kooperationsbeziehungen unterstützt werden.

- Je nach Art der Arbeitsaufgabe,
- je nach Portfolio oder Person,
- je nach Einwahl und Ausweis als Akteur,
- je nach Gruppenzugehörigkeit,
- je nach Content-Objekt-Menge bzw. Container

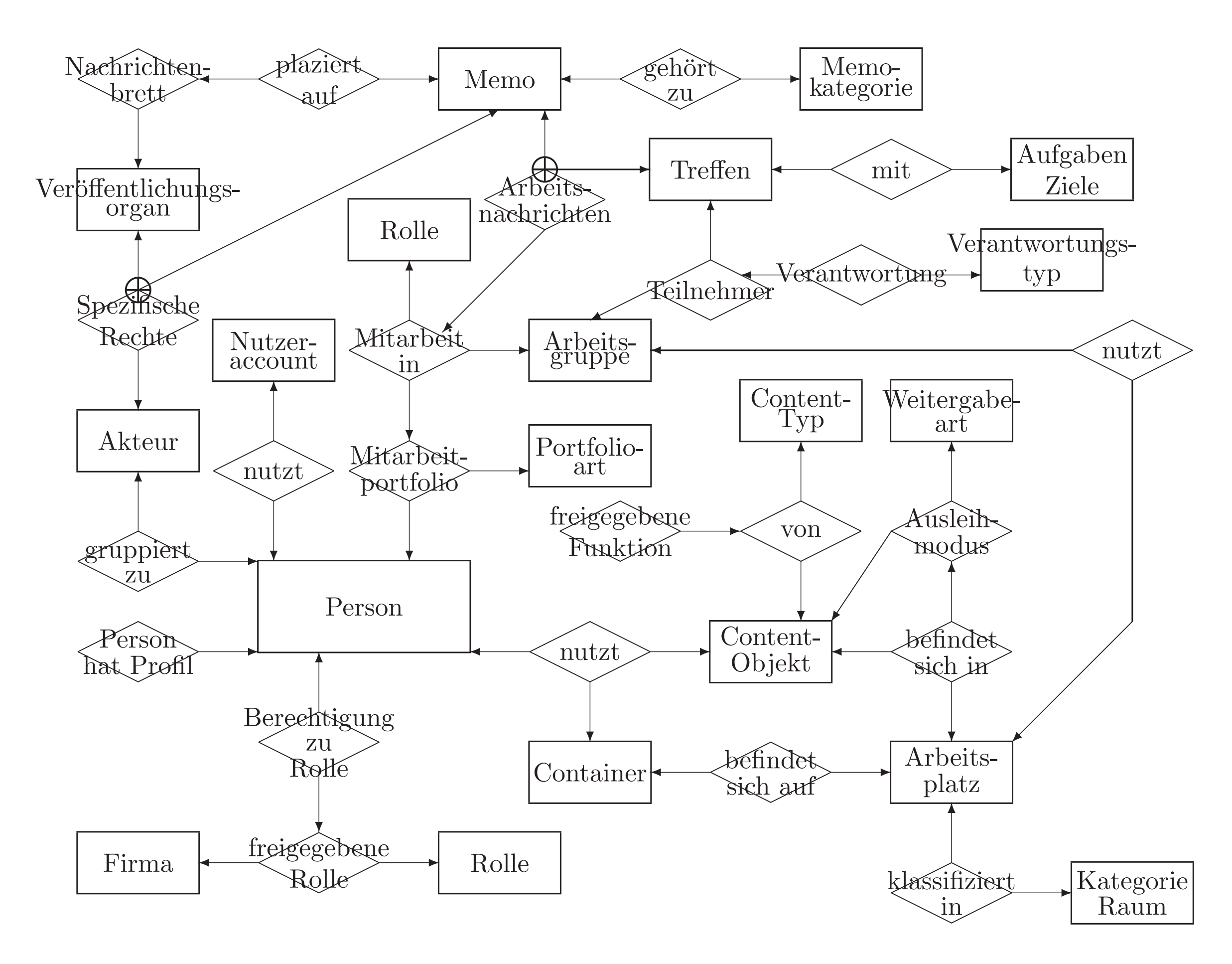


Bild 36: Teil des Schemas für den Content-Typ Arbeitsplatz (ohne Attribute und Beschränkungen)

wird einem Benutzer ein anderer Arbeitsplatz zur Verfügung stehen. Damit besitzt eine Person unterschiedliche Rechte.

Akteure sind dann z.B. der Administrator, der Leiter einer Arbeitsgruppe und der Besitzer von Content-Objekten oder Containern.

Die Außendarstellung für den anonymen Benutzer wird über das Nachrichtenbrett realisiert.

Auf dem Content-Typ Arbeitsplatz kann zur Laufzeit ein Sitzungsobjekt aufgesetzt werden. Damit dies in allgemeiner Form möglich ist, führen wir Sichten über dem Content-Typ ein, die wir Sitzungs-Schema $\mathbb{S}_{Arbeitsplatz}(Parameter)$ nennen.

Ein Sitzung-Objekt $\mathbb{SO}_{Arbeitsplatz}(a, b, ...)$ stellt eine Instantiierung des Sitzungs-Schemas und eine Einbettung in den Kontext dar.

Ein Sitzungs-Schema ist eine parametrisierte Sicht auf den Content-Typ Arbeitsplatz. Parameter eines Sitzungs-Schemas sind dann Auswahl-Objekte, mit denen die Daten und Funktionen für eine Sitzung freigeschaltet werden können.

Im Beispiel von Bild 36 ist dies ein Parameter

Aufruf = `choose` *Person (Name, Login, Paßwort)*
`or` *Mitglied (Name, Login, Paßwort, Arbeitsgruppe)*
`or` *Akteur (Name, Login, Paßwort, Rolle)*
`or` *Anonymität*

Damit werden die entsprechenden Sichten und Funktionen freigeschaltet. Gleichzeitig wird die Konsistenz in der Benutzung entsprechend der gewählten Kooperationsbeziehungen gewahrt. Damit wird ein Benutzer auf unterschiedliche Art unterstützt:

Person als zentraler Einwahlpunkt in den Arbeitsplatz: In diesem Fall werden unter Berücksichtigung der Rollen, Rechte und des Portfolios der Arbeitslatz mit den Containern und Content-Objekten aufgebaut.

Mitglied einer Arbeitsgruppe mit einer Einwahl in die Arbeitsgruppe, den für die Arbeitsgruppe freigegebenen Arbeitsplätzen, den entsprechenden Containern und den aktuellen Arbeitstreffen.

Akteur mit einer Einwahl über die Person und die Rolle, dem Freischalten von entsprechenden Teilen des Content-Objektes zur Bearbeitung von Daten etc.

Anonyme Benutzung der freigegebenen Nachrichten, Content-Objekte und allgemeinen Übersichten.

Das Content-Objekt $\mathbb{SO}_{Arbeitsplatz}(Akteur(BernhardThalheim, thalheim, ****, Arbeitsgruppenleiter))$ generiert dann z.B. die Content-Objekte, Container und Schreibtische, die der Autor auf seinen Arbeitsplatz als Arbeitsgruppenleiter besitzt.

Auf analoge Art können der Content-Typ *Persönlicher Arbeitsraum* und der Sitzungs-Typ $\mathbb{S}_{PersönlicherArbeitsraum}(Parameter)$ realisiert werden.

Damit steht eine allgemeine Technologie zur Realisierung beliebig komplexer Szenario zur Verfügung. Diese Technologie erlaubt auch die Generierung entsprechender Begleitinformation, das Aktualisieren der entsprechenden Datenbestände und kann durch Integration entsprechender allgemeiner und inhaltsbasierter Repräsentationsstile, einschließlich entsprechender Metaphern eine automatische Generierung von Arbeitsoberflächen unterstützen.

Sichtenkooperation und Integrationsschema

Das Problem der Sichtenintegration ist ein unentscheidbares Problem. Eine vollständige Sichtenintegration ist jedoch in der Praxis weder erforderlich noch erwünscht. Oft sollen Datenbestände auch lose gekoppelt bleiben. Die Theorieeinsicht, daß eine Sichtenintegration unentscheidbar ist, steht der Praxisbeobachtung gegenüber, bei der Daten in unterschiedlichen Anwendungen relativ einfach miteinander in Beziehung stehen können. Die Anwendungen verwenden allerdings in der Praxis ein explizites oder implizites *Integrationsschema*. Wir wollen diese Idee weiterverfolgen.

Eine Integration von Daten ist aus einer Vielzahl von Gründen nicht möglich. Die Sichtenintegration wird durch verschiedene Vereinbarkeitsprobleme und Konflikte erschwert:

Strukturelle Konflikte: Die Strukturen entsprechen einander nicht.

- Unterschiede in Schlüsseln: Es existieren nur verschiedene, nicht integrierbare Schlüssel.
- Abstraktionsgranularität: Die Abstraktion der verschiedenen Typen ist unterschiedlich. Zum Beispiel sind *Vorlesung* und *Kurs* in unserem Beispiel nicht auf gleichem Abstraktionsniveau.
- Verschiedene Zeitmaße und Wertebereiche: Die Repräsentation und die Wertemengen von Attribut-Typen können einander entsprechen, ohne daß dies direkt ersichtlich ist.
- Fehlende Typen: Da Sichten eine eingeschränkte Welt repräsentieren, sind sie unvollständig.

Semantische Unterschiede: Die Bedeutung bzw. Semantik der Konzepte ist unterschiedlich.

- Unterschiede im Gültigkeitsbereich: Es gelten weder Inklusions- noch Exklusions-, noch die negierten Inklusions-, noch die negierten Exklusionsabhängigkeiten.
- Wertesemantik: Die Bedeutung der Werte umfaßt zusätzliche Werte, wie z.B. die *Matrikelnummer*, die das Immatrikulationsjahr mit einschließt.
- Verschiedene Konstruktoren bei Synonymen: Verschiedene Konstruktoren für äquivalente Mengen führen zu relativ komplexen Integritätsbedingungen, die in den Sichten fehlen.

Verschiedene Operationen: Auf den Typen sind unterschiedliche Operationen definiert, die nicht integrierbar sind.

Verschiedene Wertebereiche: Synonyme Typen besitzen verschiedene Wertebereiche.

Wir können jedoch mit dem ER-Modell Mechanismen bereitstellen, die eine **Kooperation** von Sichten unterstützen.
Gegeben seien die Sichtenschemata $\mathfrak{S}_1$ und $\mathfrak{S}_2$ und entsprechende Datenbanken $\mathfrak{S}_1^C$ und $\mathfrak{S}_2^C$.

Ein partieller *Schema-Morphismus* von $\mathfrak{S}_1$ und $\mathfrak{S}_2$ wird

- als Paar (f_{12}, f_{12}^C) von Funktionen definiert, so daß
- f_{12} eine partielle Einbettungsfunktion von Typen von $\mathfrak{S}_1$ in Typen von $\mathfrak{S}_2$ mit dem Definitionsbereich $def(f_{12}) = \mathfrak{S}_{12}$ ist,
- f_{12}^C ist die korrespondierende Einbettung von Klassen von $\mathfrak{S}_1^C$ in die von $\mathfrak{S}_2^C$, d.h.
  - $f_{12} : \mathfrak{S}_1 \not\rightarrow \mathfrak{S}_2$ und
  - $f_{12}^C : \mathfrak{S}_1^C \not\rightarrow \mathfrak{S}_2^C$
  - mit der Eigenschaft $f_{12}^C(\mathfrak{S}_1^C) \models_T \mathfrak{S}_2 |_{f_{12}(T)}$ falls f_{12} definiert ist für den Typen T, d.h. die Abbildung f_{12} erhält die Semantik von $\mathfrak{S}_2$.

Damit kommutiert das linke Diagramm in Bild 37 .

Die Partialität von (f_{12}, f_{12}^C) definiert ein Sichtenschema $\mathfrak{S}_{11}$ in $\mathfrak{S}_1$ und eine Sicht $f_{12}(\mathfrak{S}_{11})$ in $\mathfrak{S}_2$. Es sei außerdem ein partieller Schema-Morphismus (f_{21}, f_{21}^C) von $\mathfrak{S}_1$ und $\mathfrak{S}_2$ gegeben.

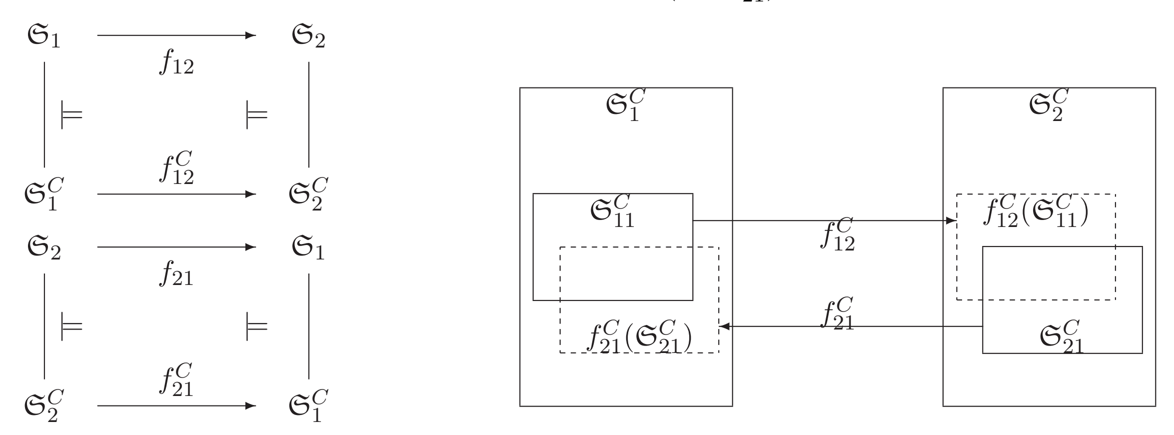


Bild 37: Partielle Schema-Morphismen zur Sichtenkooperation

Die Schema-Morphismen (f_{12}, f_{12}^C) und (f_{21}, f_{21}^C) definieren eine Sichtenkooperation falls

- für jeden Typ $T_1 \in \mathfrak{S}_{11} \cap f_{21}(\mathfrak{S}_{21})$ und jedem Typ $T_2 \in \mathfrak{S}_{21} \cap f_{12}(\mathfrak{S}_{11})$,
- für jedes Paar der entsprechenden Klassen $T_1^C \in \mathfrak{S}_1^C$, $T_2^C \in \mathfrak{S}_2^C$
- die Funktionen $f_{12}^C(T_1^C), f_{21}^C(f_{12}^C(T_1^C)), f_{21}^C(T_2^C), f_{12}^C(f_{21}^C(T_2^C))$ definiert sind und
- die kommutierenden Gleichungen $f_{21}^C(f_{12}^C(T_1^C)) = T_1^C$, $f_{12}^C(f_{21}^C(T_2^C)) = T_2^C$ gelten.

Durch die Sichtenkooperation wird ein Input eines Schemas mit dem Output eines anderen Schemas verknüpft. Diese Verknüpfung erlaubt eine Verbindung von Sichten, wie in Bild 37 bei Angabe der Funktionen f_{12} und f_{21}.

Sind die Typen der Sichten entweder über Schema-Morphismen total verbunden oder paarweise verschieden, dann sprechen wir von der Sichtenintegration. Eine Sichtenintegration können wir damit formal definieren. Meist wird eine Sichtenintegration nur pauschal und informal in der Literatur eingeführt. Mit dem Schema-Morphismus können wir die Sichtenintegration auch formal fassen. In diesem Fall gelten:

- $f_{12}(\mathfrak{S}_{11}) = \mathfrak{S}_{21}$ und
- $f_{21}(\mathfrak{S}_{21}) = \mathfrak{S}_{11}$.

Sind zwei Typen total verbunden, wird einer der Typen der Schemata zur Weiterführung im integrierten Schema ausgewählt und der nicht verwendete Typ über eine Sichtendefinition an den verbleibenden Typen gebunden.

Die Assoziierbarkeit von Typen der Schemata wird durch die Wertebereiche der Typen der Sichtenschemat begrenzt:
Typen T_1 in $\mathfrak{S}_1$ und T_2 in $\mathfrak{S}_2$ sind *wertebereichsverträglich*, falls $dom(T_1) = dom(T_2)$ gilt.

Es ist anzumerken, daß die Wertebereichsverträglichkeit nicht auf eine Teiltypen-Eigenschaft $\mathfrak{S}_{11} \cap f_{21}(\mathfrak{S}_{21})$ und in $\mathfrak{S}_{21} \cap f_{12}(\mathfrak{S}_{12})$ für die Morphismen der Sichtenkooperation reduzierbar ist.

Die beiden Schema-Morphismen (f_{12}, f_{12}^C) und (f_{21}, f_{21}^C) definieren eine Gleichungstheorie $\mathfrak{E}$. Wir können vereinfachend annehmen, daß alle Typen-Namen in den Schemata $\mathfrak{S}_1$ und $\mathfrak{S}_2$ verschieden sind. Damit können wir für alle Typen T_1 in $\mathfrak{S}_1$ die Gleichung $T_1 = f_{12}(T_1)$ und für alle Typen T_2 in $O_2^{\mathcal{V}}$ die Gleichung $T_2 = f_{21}(T_2)$ zur Gleichungstheorie $\mathfrak{E}$ hinzufügen.

Falls wir an einer vollständigen Integration interessiert sind, dann können die Gleichungen durch *Term-Ersetzungsregeln* der Form $T \rightsquigarrow f_{ij}(T)$ oder $f_{ij}(T) \rightsquigarrow T$ ersetzt werden. Diese Ersetzungsregeln müssen auch dem induktiven Aufbau der Typen folgen. Deshalb wird auch ein Ableitungssystem benötigt. Wir nutzen dazu die folgenden Inferenzregeln:

$$\frac{\mathfrak{E} \cup \{T(T_1, ..., T_m) = S(S_1, ..., S_m)\}}{\mathfrak{E} \cup \{T_1 = S_1, ..., T_m = S_m\}} \qquad \frac{\mathfrak{E} \cup \{T = T\}}{\mathfrak{E}} \qquad \frac{\mathfrak{E} \cup \{T = S\}}{\mathfrak{E} \cup \{S = T\}} \qquad \frac{\mathfrak{E} \cup \{T = S\}}{\vartheta_{T \rightsquigarrow S}(\mathfrak{E}) \cup \{T = S\}}$$

für Substitution von $\vartheta_{T \rightsquigarrow S}$ in $\mathfrak{E}$.

Zwei Sichtenschemata sind integrierbar, wenn Schema-Morphismen existieren, die alle Typen der Schemata paarweise miteinander assoziieren.

Wir können die Sichtenintegration auch durch Definition entsprechender Anfragemengen definieren. Diese Definition ist der obigen äquivalent.

7 Sprachen zur Darstellung der Interaktivität

Greift nur hinein ins volle Menschenleben!
Ein jeder lebts, nicht vielen ists bekannt,
Und wo Ihrs packt, da ists interessant.
In bunten Bildern wenig Klarheit:
So wird der beste Trunk gebraut,
Der alle Welt erquickt und auferbaut.
Goethe, Faust, Erster Teil, Vorspiel auf dem Theater, Lustige Person

Der Interaktionsraum zur Darstellung der Interaktivität

Die Interaktivität kann unter zwei Gesichtspunkten dargestellt werden:

Der System-Gesichtspunkt umfaßt alle Input-, Output- und Speicherprozesse und baut auf der Strukturierung der Daten, auf den Sichten zur zusammenhängenden Darstellung der Daten, sowie auf dem technischen Workflow, der wiederum auf Systemprozessen basiert, auf.

Der Benutzer-Gesichtspunkt basiert auf den Rollen und Aufgaben von Benutzergruppen, deren Sichtweisen auf die dargestellten Daten und die ablaufenden Prozesse. Diese Sichtweisen sind auch durch die Pragmatik der Benutzergruppen geprägt.

Ein Informationssystem basiert auf einer Schichten-Architektur, die die klassische ANSI-Sparc-Architektur verallgemeinert. Im folgenden vertiefen wir diesen Zugang. Die Architektur ist in Bild 38 (b) skizziert. Mit dieser Architektur wird nicht nur die klassische Seeheim-Architektur in Bild 38 (a) verbessert, sondern auch eine ganzheitliche Betrachtung von Anwendungen ermöglicht. Die Oberflächenmodellierung wurde auch für Datenbanken im wesentlichen auf der Grundlage des Seeheim-Modelles nach Bild 38 (a) (ohne Dialogmanagementsystem und Sichtengenerator) vorgenommen. Das klassische Seeheim-Modell trennt wie in Client/Server-Architekturen die Präsentation vom Anwendungssystem. Diese Trennung hat sich für eine Vielzahl von Anwendungen durchgesetzt. Die Funktionalität der Anwendungssysteme kann sich dabei weiter in die Clients verlagern. Für Datenbanksysteme hat sich diese Architektur sogar mit einer Verallgemeinerung zur Arch-Architektur noch nicht durchgesetzt. Vorstellbar ist nach [Sch96] auch eine Erweiterung der Präsentationskomponente zu einem Dialogmanagementsystem. Die Arbeiten der DBIS-Arbeitsgruppe haben zu der hier verwendeten verallgemeinerten Architektur geführt.

Die Trennung zwischen Client und Server ist eine der möglichen Separation innerhalb einer Anwendung. Vorstellbar sind weitere Trennungen, wie z.B. die Trennung für verteilte Informationssysteme, die Trennung für Web-Informationssysteme mit relativ einfachen Client oder auch Applet-basierte Clients. Das DBIS-Modell ist auf keine der Trennlinien angewiesen und erlaubt eine spätere Entscheidung für eine Plattform.

Typische weitere Trennungen sind meist als Multi-Tier-Architekturen, z.B. als 3-Tier-Architekturen spezifiziert.

Die Spezifikation des Interaktionsraumes wird in folgenden Entwurfsdokumenten niedergelegt:

Drehbuch: Der Ablauf der Interaktion, die Akteure, die Stories der Anwendung werden im Drehbuch zusammengefaßt.

Content-Typen: Das Systeminterface wird als Container-Objekt bereitgestellt, mit dem ein Akteur sowohl die aktuellen und spezifischen Sichtweisen auf die Datenbank erhält, als auch die entsprechende Funktionalität zum Agieren mit dem Informationssystem.

Der Interaktionsraum wird um "Soft"-Bestandteile erweitert:

Kollaborationsrahmen: Die Interaktion basiert auf der Existenz mehrerer Parteien, die in unterschiedlichen Rollen agieren, kollaborieren und unterschiedliche Interessen verfolgen.

Das verallgemeinerte Seeheim-Modell **Das DBIS-Modell für Informationssysteme**

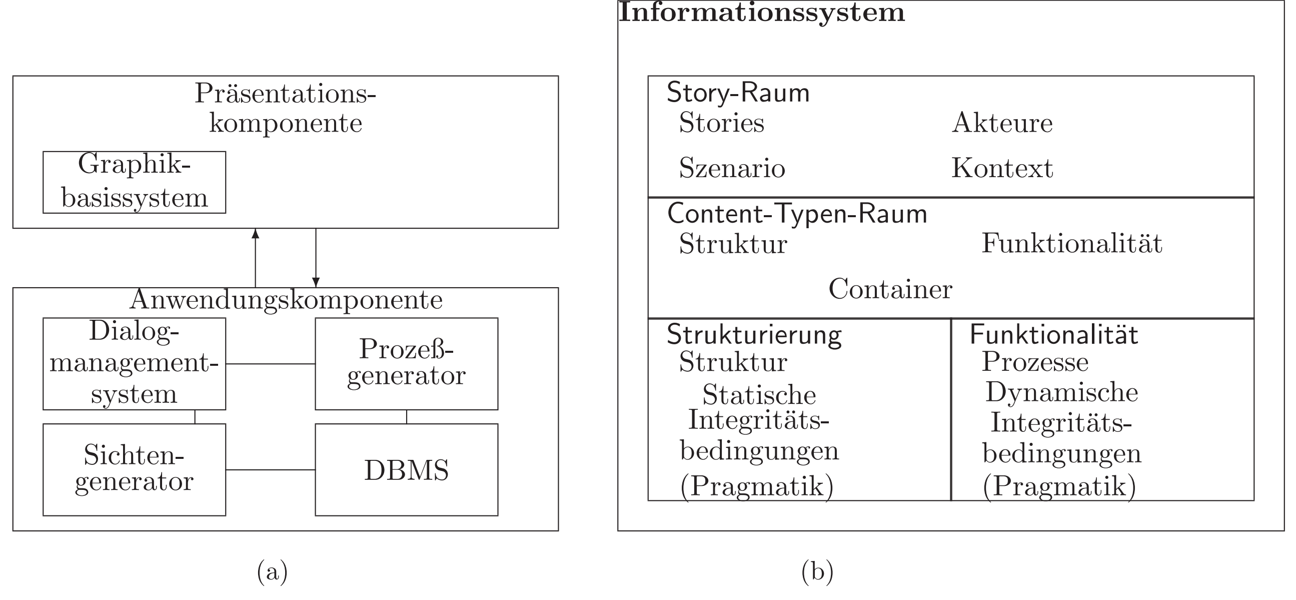


Bild 38: Spezifikation von Informationssystemen

Gestaltungsrahmen: Bei der Gestaltung von Benutzungsschnittstellen ist es angebracht, einem einheitlichen Schema zu folgen. Wir fassen den "Style Guide" zur Gestaltung von Interfaces, die Metaphorik und die allgemeinen Gestaltungsrichtlinien im Gestaltungsrahmen zusammen.

Arbeitsrahmen: Informationssysteme sollen bei der Bewältigung von Arbeitsaufgaben eingesetzt werden. Deshalb müssen auch das Portfolio, das Aufgabenspektrum der einzelnen Benutzer und die Lösungsschritte für die Arbeitsaufgaben angemessen bei der Gestaltung berücksichtigt werden.

Interaktivität im Abstraktionsschichtenmodell

Wie bereits in den vorhergehenden Teilen diskutiert, unterscheiden wir zwischen *dem Diskurs*, den *Handlungsrahmen*, dem *Storyboard*, dem *Drehbuch* und der *Inszenierung* der Dialoge. In unterschiedlichen Entwurfsetappen werden die Dialoge im Abstraktionsschichtenmodell spezifiziert. Informationssysteme sind meist auf unterschiedliche Benutzergruppen ausgerichtet, die unterschiedliche Anforderungen an die Benutzung, an das intuitive Verständnis der einzelnen Schritte, an die Funktionalität und die Gestaltung der Oberflächen haben. Da eine zusammenhängende Darstellung nach unserer Kenntnis nicht existiert, stellen wir unsere Methodik ausführlicher vor.

Das Finden der Motive und Ideen und die Darstellung des Diskurses kann auf den Informationen, die wir bereits in der Anwendungsanalyse erhalten haben, aufsetzen. Wir entwickeln erste grobe und bruchstückhafte Ideen. Später können wir aus diesen Ideen eine Auswahl treffen. In dieser Etappe ist eher eine Methode wie das *mind mapping* angebracht. Damit ist ein Entwerfer voll gefordert. Oft ist nicht die objektivste Auswahl von Ideen die beste, sondern eine subjektive Auswahl. Dabei zeigt sich, daß das Ideenmaterial eigene Prinzipien hat und auch widersprüchlich sein kann. Es wird in diesem Schritt das *Anwendungsgebiet* mit den einzelnen *Anwendungsschritten* skizziert. Das Ergebnis ist die Darstellung des Diskurses im Lastenheft.

Die Entwicklung des Handlungsrahmens kann nun zu einer groben Darstellung der Aktionen der Akteure führen. Wir modellieren deshalb die Akteure in dieser Etappe mit ihren Rollen, Rechten, Aufgaben und Zielen im Groben. Der Handlungsrahmen ist mit der Darstellung der Motive und Ziele im vorigen Schritt bereits skizziert.

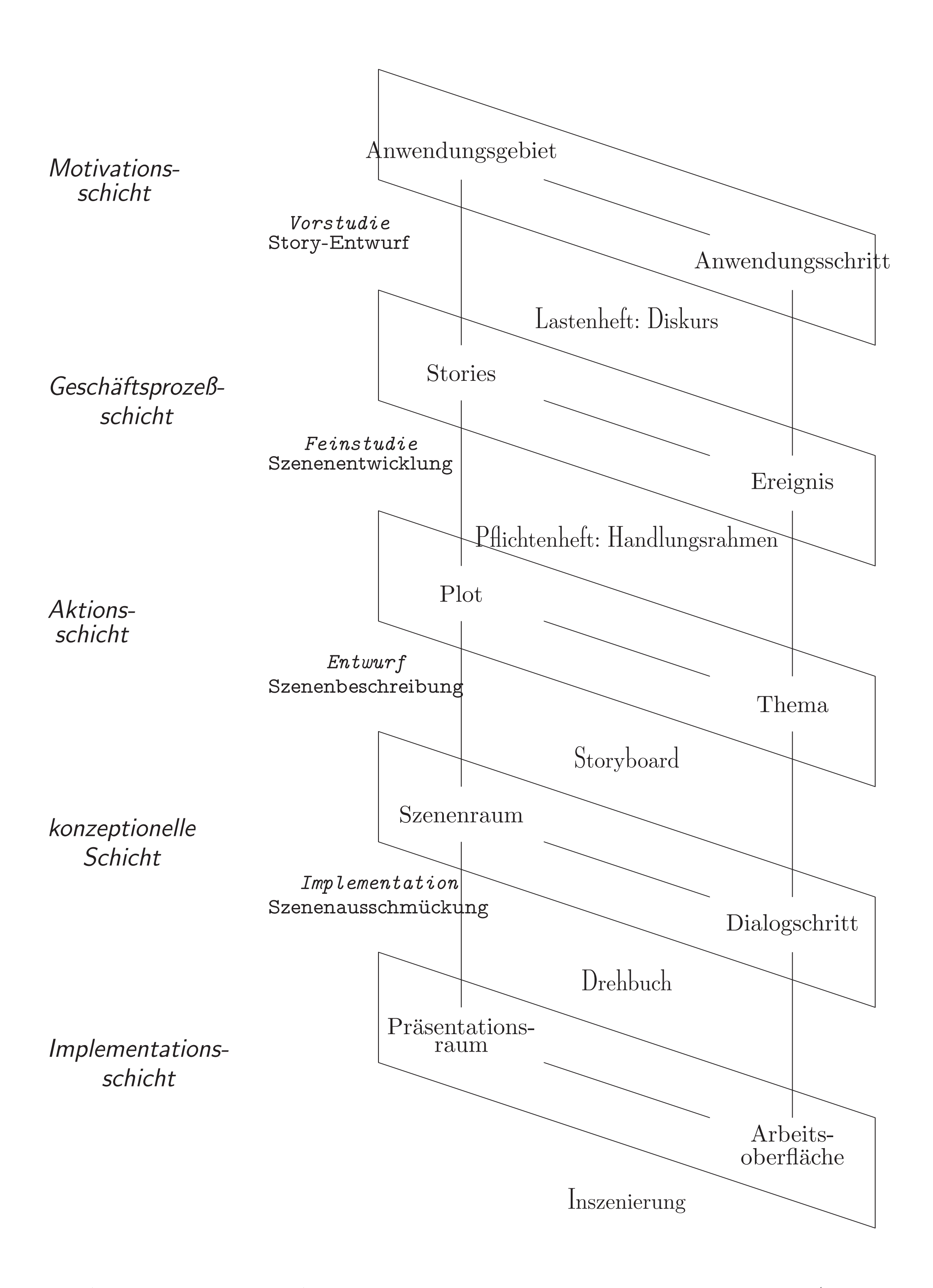


Bild 39: Die Arbeitsprodukte im Abstraktionsschichtenmodell für den Story-Raum (Dialogaspekte)

worin die Geschichte besteht. In der Geschichte werden die Hauptdialoge mit ihren Zielen und Absichten dargestellt. Nicht alle Einzelszenen müssen enthalten sein.

Es existiert eine Vielfalt von möglichen Stories. Trotzdem gibt es Regeln zur Beschreibung von Geschichten. Jede Geschichte wird durch Motive, Absichten und Ziele geprägt. Damit ist auch ein Skelett der Handlung gegeben. Auf der Grundlage dieses Skeletts kann die Geschichte eine Struktur erhalten. Sie sollte frei von Widersprüchen und nur beschränkt rekursiv sein.

Ein System wird nur dann akzeptiert, wenn es einen intuitiv erkennbaren Nutzen bringt und echte Bedürfnisse von Akteuren in einfacher Form befriedigt. Ein System ist damit auch vom Zeitgeist abhängig, sollte sich diesem aber nicht vordergründig verpflichtet fühlen. Jede Szene ist klar und deutlich zu entwerfen und muß mit einem entsprechenden Inhalt an der richtigen Stelle, mit der richtigen Hintergrundinformation und mit adäquaten Aufgaben komponiert werden. Außerdem sind für jede Szene die Informationen den Akteuren *in der richtigen Sorte*, *in der richtigen Dosis*, *in der richtigen Form*, *in vollem Umfang* und *zu akzeptablen Kosten* zur Verfügung zu stellen. Allen Akteuren ist klar und deutlich darzustellen, worin der nächste Arbeitsschritt besteht, in welcher Szene der Story er sich befindet und welche Probleme nun gelöst werden sollen und können.

Eine Anwendung kann auf eine Fülle von Zielgruppen oder auf einige wenige Akteure orientiert werden. Anstatt eine Story 'drauflos zu entwickeln', bevorzugen wir eine methodische Entwicklung. Wir arbeiten uns von der Idee zur Grobstruktur und weiter über verschiedene Zwischenstadien bis zur Endfassung vor. Die drei wichtigsten Entwicklungsstadien sind *Expose*, *Treatment* und die *ausgearbeite Story*. Ein solches schrittweises Vorgehen bringt beträchtliche Vorteile durch die schrittweise Beseitigung von Unsicherheitsfaktoren und das Hinzufügen von zusätzlichem Material mit sich. Jede Szene kann damit ihren richtigen Platz in der Story erhalten. Sprünge werden vermieden. Der langsame Aufbau gewährleistet auch Detailtreue.

Eine Story baut auf *Ereignissen* auf, in denen Akteure in Arbeitsschritten ontologische Einheiten benutzen. Deshalb wird hier auch eine enge Integration der Dialogentwicklung mit der Entwicklung der Sichten und der Funktionen vorgenommen.

Das Resultat dieses Schrittes ist als *Handlungsrahmen* Bestandteil des *Pflichtenheftes*.

Die Spezifikation des Storyboards wird auf der Grundlage der entwickelten Story durch Auswahl von möglichen Ausprägungen und Verfeinerung entwickelt. Die Story besteht aus Szenen, die nun in einer Form ausgeprägt werden, die dem tatsächlichen Ablauf der Handlung entspricht. Wir nutzen dazu eine Aufnahme der möglichen Szenario. Ein Szenario ist ein genereller Ablauf aus der Sicht der Akteure. Dieser Auflauf oder Durchlauf soll dem aktuellen Geschehen in der Anwendung entsprechen.

Die einzelnen Szenario können wir schrittweise miteinander verknüpfen und diese integrieren. Mit einer derartigen Integration entsteht eine Verfeinerung der Story. Die einzelnen Szenen werden nun durch entsprechende Dialogschritte untersetzt, in denen die Akteure entsprechende Handlungen und Aktionen vornehmen und dazu Daten vom Format der Aktionssichten-Suite verwenden.

Zwischen Story und Szenarien existiert ein Unterschied. Die Geschichte ist das eigentliche Geschehen. Die Szenario bestimmen die *Auswahl* von Szenen und Szenenfolgen. Jede einzelne Szene stellt ein Thema der Anwendung dar. Im Falle unserer Beispielanwendung sind Themen *Angabe von Vorschlägen zu Lehrveranstaltungen*, *Zusammenstellung eines Stundenplanes*, *Übersicht über ein Institutsprofil*.

Die Szenario stellen einen verfeinerten Ablauf einer einzelnen Story dar. Dabei wird es oft vorkommen, daß nicht eine einzelne Story zur Darstellung aller möglichen Szenario ausreicht, sondern eine Menge von Stories, die die Abläufe in der Anwendung beschreibt. In diesen Fall entwickeln wir den Raum der Stories, den *Story-Raum*. Dieser Story-Raum kann auch auf andere Art durchlaufen werden als in den angegebenen Szenario. In diesem Fall entdecken wir Lücken in der Darstellung der Anwendung. Die Stories werden durch einen Plot in diesem Entwurfsschritt verfeinert. Das Plot ist *eine Anordnung* der Ereignisse des Story-Raumes. In der Dramaturgie

(Film, Drama, Erzählung, Musik) wird oft nur eine einzelne Story zur Grundlage genommen. In der Architektur sind Plots nichtlinear. Plots umfassen

- die *Raumplanung* und die *Raumordnung* für die Stories, d.h. die Planung und den Ablauf der Szenen,
- den allgemeinen *Ablauf der Themen*,
- Prinzipien zur *Szenographie* und zum *Aktionsraum*,
- die Aktionen der *Darsteller und Akteure*,
- Prinzipien der *Komposition und des Klangbildes*, die als Qualitätsparameter dargestellt werden können, und
- Prinzipien der *Akzeptanz und Aufnahme* (in der Dramaturgie der Musik: Melodie und Rhythmus).

Es ist offensichtlich, daß nicht alle Plots in der gleichen Form dargestellt werden können. Die Plots werden für die Ausarbeitung der Szenario aufbereitet. Das vorhandene Material wird auf eine einfache und klare Handlungsfolge reduziert. Die Story wird damit konkretisiert bzw. verfeinert. In der Story sind keine detailliert ausgearbeiteten Szenen enthalten, dies trifft auch für das Szenario zu. Es enthält die Szenenabfolge und alle Dialoge. In das Szenario fließt bereits die gesamte Informationsfülle ein. Sobald wir uns für eine bestimmte Auswahl entschieden haben, kommen neue Informationen hinzu. Sie ergeben sich aus dem bisher Betrachteten. Damit 'entwickelt sich das Szenario selbst'. Es werden auch Unzulänglichkeiten und Fehler sichtbar.

Die einzelnen Szenen kann man sich durch überlappende Blöcke darstellen. Da eine Information und eine Aktion noch nachwirken kann bzw. antizipiert wird, sind die Szenen nicht vollständig trennbar.

Mit der Szenenentwicklung betten wir auch die Dialoge in die Handlungen und die Daten ein. Handlungen sind Folgen von Aktionen. Aktionen benötigen Daten als benutztes Wissen, für die Ein- und Ausgabe. Eine Sicht entspricht dann einer oder mehreren Aktionen. Damit wird für die Szenarien auch die Darstellung von *Motiv, Absicht* und *Ziel* weiter verfeinert.

Ein Motiv kann zu einer Absicht führen. Einer Absicht liegt gewöhnlich ein Wunsch zugrunde, ein bestimmtes Ziel zu erreichen. Jede Aktion führt zu einem (meist erwünschten) Ergebnis. Hinter jeder Absicht steckt ein Ziel. Keine Aktion erfolgt ohne Grund. Das Motiv ist die Ursache der Aktion. Zwischen Ursache und Wirkung besteht eine direkte Verbindung.

Absichten haben verschiedene Eigenschaften, sind direkt, indirekt, bewußt, unbewußt, freiwillig, unfreiwillig, offensichtlich oder versteckt. Kann eine Absicht nicht verwirklicht werden, entsteht ein Konflikt oder evt. auch nur ein Gegensatz.

Das Ziel orientiert auf ein in der Zukunft liegendes Ereignis, das durch eine Absicht herbeigeführt werden soll. Beide Kategorien können beliebig weit auseinander liegen.

Zwischen den Aktionen gibt es Verknüpfungspunkte. Absichten können auch von einer Gruppe von Akteuren bzw. von Akteuren mit verschiedenen Rollen gleichzeitig getragen werden.

Ein Szenario muß auch akzeptabel sein. Damit werden Benutzerbedürfnisse anhand der Spezifikation des Szenarios geprüft. Dabei konzentrieren wir uns auf folgende Probleme:

Verständlichkeit: Jedes Szenario und jede Szene muß verstanden werden. Deshalb ist Klarheit und Verständlichkeit oberstes Gebot, wobei die Inhalte für alle Anwender (ggf. auch weltweit) die gleiche Semantik besitzen müssen. Der Benutzer kann nur entsprechend seinen Erfahrungen fehlende Teile antizipieren. Er soll vom Motiv auf die Absicht und von der Absicht auf das Ziel schließen können. Sind wesentliche Teile unverständlich, dann kann er keine Schlußfolgerungen ziehen. Der Benutzer will Informationen, die er noch nicht kennt, d.h. es werden neue Informationen geliefert, die sich anhand des Allgemeinwissens einordnen lassen. Bei der Vermittlung von z.T. komplexen und tiefgründigen Inhalten ist besondere Sorgfalt bei der Ausschöpfung aller Darstellungsmöglichkeiten notwendig.

Plausibilität: Ein Szenario muß plausibel sein und sollte sich an die gewohnten Arbeitsweisen anlehnen. Der Stellenwert der Plausibilität und des Realismus ist dabei umgekehrt proportional zum Auffassungsvermögen und Ausbildungsgrad.

Identifikation: Mit einem Szenario muß auch das Interesse der Akteure geweckt und wach gehalten werden. Für die Akteure muß eine enge Verflechtung zwischen dem Inhalt, den Prozessen und den Dialogen einerseits und der Arbeitsweise anderseits erreicht werden. Ein Benutzer soll sich mit 'seinem' System identifizieren können. Die Identifikation hat eine ganze Reihe von erwünschten Auswirkungen und ist ein wesentlicher Grund für die Akzeptanz eines Systemes.

Für das Szenario bewerten wir abschließend seine *Qualität*.

Vollständigkeit: Alle Szenen sind vollständig und bis ins Detail ausgestaltet.

Bedürfnisgerecht: Die Aktionen, Informationen und Dialoge entsprechen den Bedürfnissen der Akteure.

Didaktisch aufbereitete Granulierung: Informationen können in der Granulierung auch einen Akteur überfordern, was häufig bei einer direkten Übertragung von Darstellungen mit Printmedien vorkommt.

Inhaltliche Konsistenz: Jede Aktion, jede Information, jeder Dialog besitzt einen lokalen und einen globalen Kontext. In beiden sollten Widersprüche vermieden werden.

Resultat dieses Schrittes ist ein *Storyboard* mit einer detaillierten Beschreibung der Szenen. Es werden die Stories aus dem Pflichtenheft mit den Ereignissen durch Plots und Themen verfeinert. Diese Szenen wiederum werden schrittweise untersetzt durch einzelne Aktionen der Akteure. Das Storyboard zeigt die Anwendung aus der Sicht des Benutzers. Oft werden dazu auch graphische Repräsentationen benutzt. Eine solche Repräsentation wird bei Website-Entwicklungen Mockup genannt. Ein Mockup stellt eine Folge oder allgemein partiell geordnete Menge von Themen dar unter Einbezug der Gestaltungsmittel der visuellen Gestaltung.

Für das Drehbuch werden die Szenen des Storyboards weiterentwickelt. Durch das Drehbuch wird die *Art und Weise*, in der die Geschichte realisiert werden soll, spezifiziert. Ein Drehbuch spezifiziert eine Story bzw. den gesamten Story-Raum im Detail. Das Drehbuch ist eine konzeptionelle Repräsentation des Handlungsablaufes aller Facetten der Anwendung.

Das Drehbuch basiert auf *Szenen*, die miteinander durch explizite Übergänge verbunden sind. Die Szenen selbst realisieren entsprechende Aktionen von Akteuren, die durch *Dialogschritte* dargestellt werden. Diese Aktionen können durch kurze prägnante Beschreibungen charakterisiert werden. Wir streben dazu auch eine Kurzcharakterisierung an. Dazu benutzen wir Verben oder auch Substantive. Diese Worte werden als *Wortfelder* dargestellt.

Eine Szene ist dann ein algebraischer Ausdruck von Dialogschritten. Die Algebra muß dazu auch die Parallelität von Schritten berücksichtigen. Einer Szene sind Akteure mit entsprechenden Rollen und Aufgaben zugeordnet. Eine Szene nutzt ein Medien-Objekt. Ein Dialogschritt wird unter Beteiligung einiger Akteure, die in die Szene involviert sind, ausgeführt. Dabei werden die Akteure einem *Kontext* zugeordnet. Dieser Kontext stellt insbesondere die technischen Rahmenbedingungen der Benutzung dar.

Wir berücksichtigen für das Drehbuch auch Eigenschaften und Wirkungen auf den Benutzer. Damit wird das Drehbuch im wesentlichen von drei Faktoren bestimmt: *von der Form, den Aktionen der Story und den Besonderheiten der Arbeitsweise der Endbenutzer*. Wir können damit im einzelnen Folgearbeitspakete herausstellen:

Kategorisierung der Endbenutzer: Aktionen und Dialoge existieren nicht unabhängig von den Akteuren. Es können die Akteure *kategorisiert* und mit *Charakteristika* versehen werden. Dabei interessieren nur solche Details, die für den Verlauf der Dialoge von Bedeutung sind, d.h. wir erfassen einige Charakteristika und charakterisieren nicht etwa den Endan-

wie notwendig mit erfaßt. Die Kategorisierung sollte sich durch eine relative Beständigkeit auszeichnen.

Aktionsphasen: So wie ein Verb Aktion bzw. Handlung verdeutlicht, kann auch Aktion in den drei Zeitdimensionen dargestellt werden, die untrennbar miteinander verbunden sind. Eine für die Zukunft geplante Aktion weist auf ein bevorstehendes Ereignis. Ihr geht eine Absicht und damit ein Motiv voraus, die sich einem allgemeinen Plan des Szenarios unterordnet. Ereignisse, die in der Vergangenheit stattfanden, sind in ihrem Zeitbezug auch darzustellen.

Aktive und inaktive Zustände: Szenen können, aber müssen nicht zu einer Aktivierung führen. Deshalb kann auch ein Szenario vorsehen, daß einzelne Aktionen 'schlummern'. Wir können diese Verzögerung durch lang andauernde Transaktionen darstellen. Die Implementierung wird damit jedoch komplexer. Bei inaktiven Zuständen fehlt ein Motiv für eine Aktion oder es liegt eine Störung vor. Zur Spezifikation ziehen wir deshalb auch die Kategorisierung der Endbenutzer mit heran. Wenden Benutzer Aktionen an, ohne agieren zu können, dann liegt ein Konflikt vor. Ein Beispiel dafür ist das exklusive Schreiben von Daten für höchstens einen Benutzer. Dazu benötigen wir eine Konfliktlösungsstrategie je nach Intensität der Absicht und unterscheiden Hindernisse von Komplikationen und diese von Gegenabsichten. Unkritisch sind dagegen inaktive Zustände, die nach Erreichen eines Zieles erreicht wurden.

Hauptabsichten und Teilabsichten: Gewöhnlich ist ein Geschäftsprozeß bzw. ein Szenario keine Kette von Ereignissen. Wir finden ein Netzwerk von Motiven, Absichten und Zielen vor. Die Absichten können in Teilabsichten, die den Hauptabsichten dienen, und Hauptabsichten kategorisiert werden. Damit ergibt sich auch eine zeitliche Ordnung und eine Variation in der Reihenfolge. Dabei können die Absichten gemeinsam dargestellt werden, die sich einem gemeinsamen Zweck unterwerfen (Gesetz von der Einheit des Zwecks). Teilabsichten sind Änderungen unterworfen, Hauptabsichten dagegen nicht. Teilabsichten sollten stets beendbar sein. Sie besitzen auch Hilfsziele.

Wirkungen auf den Benutzer: Die Art und Weise, wie verschiedene Kategorien von Benutzern in ihren Rollen auf Ereignisse reagieren, wird in die Gestaltung des Drehbuches mit einbezogen. Wir untersuchen dazu für die Benutzergruppen auch die Antizipationsfähigkeit, den Erfahrungsschatz und die Fähigkeiten zur Bewältigung von Schwierigkeiten und benutzen für die Gestaltung der Szenen diese Informationen. Eine kluge und durchdachte Ereignisstruktur ist Voraussetzung für eine Akzeptanz der Dialoge. Der Benutzer soll in der Lage sein, die Distanz zum Erreichen des Ziels abzuschätzen, wozu auch eine Umstellung der Szenen beitragen kann.

Eigenschaften der Darstellungsmedien: Der Entwickler kann sich vieler Medien bedienen, um alle Bestandteile einer Szene dem Akteur mitzuteilen. Es müssen Dialoge, Geräusche, Handlungen, Dekorationen, Darstellungsobjekte und Musik in konsistenter Form eingesetzt werden.

Damit ist das Verfassen ebenso wie alle anderen Schritte der Entwurfsschicht nicht nur zu eine schöpferischen Tätigkeit, sondern ist vor allem auch ein *Handwerk*, das sich an Regeln der Handwerkskunst orientiert. Am Ende entsteht auf der Grundlage des Szenarios, der Story und der Ideen ein ausgereiftes Drehbuch.

Die Szenenfolge wird anschließend auf Variation, Veränderung und Kontrast untersucht. Für die einzelnen Szenen sind die *Verbindungen* explizit zu modellieren. Deshalb werden evt. zusätzliche Verbindungselemente aufgenommen. Nicht alle Szenen sind miteinander gleich eng verbunden. Es lassen sich *Szenenblöcke* (Akte) mit besonders starken Verbindungselementen bilden.

Nach der Fertigstellung des Drehbuches sollte man den Entwicklungsprozeß umkehren und eine *Zusammenfassung* des vorliegenden Drehbuches schreiben. Diese Zusammenfassung ist dann mit dem Storyboard, dem Story-Raum und den Ideen und Motiven zu vergleichen.

Für das Drehbuch bewerten wir abschließend seine *Qualität*, d.h. insbesondere die folgenden Qualitätskriterien:

Benutzerführung: Die Akteure benötigen neben einer angepaßten Hilfe auch eine Führung durch komplexe Prozesse.

Differenzierung: Es werden die unterschiedlichen Kategorien von Akteuren unterstützt.

Medienmix: Die Medienauswahl erfolgt an den Inhalt angepaßt.

Hypermediale Struktur: Es werden die Aktionen, Informationen und Dialoge in einer benutzergerechten Form geboten, die ein Verlieren im 'Cyberspace' verhindert.

Verbindung von Arbeit und Vergnügen: Da eine multimediale Darstellung auch eine 'Emotionalisierung' der Darstellung erlaubt, ist der Einsatz dieser Mittel zu konzipieren.

Konsistenz: Jede Szene muß an sich und auch in ihrem Kontext konsistent sein.

Erwartungskonformität: Die Erwartungen der Akteure sind für unterschiedliche Szenen verschieden. Dabei sind auch verschiedene Kategorien von Akteuren zu beachten.

Für die Inszenierung wird die Form der *Dialoge* und damit der *Präsentationsraum* bestimmt. Wir entwickeln in dieser Schicht die Arbeitsoberflächen für jeden Dialogschritt im einzelnen. Ebenso wie das Storyboard den Handlungsrahmen nicht verändert, wird durch die Inszenierung das Drehbuch nicht verändert. Es werden die einzelnen Szenen und Dialogschritte des Szenenraumes ausgeschmückt.

Die Spezifikation der Dialogschritte im Drehbuch basiert bereits auf einem Rahmen. Wir können diesen Rahmen als Start für die Spezifikation eines *Gestaltungsrahmens* oder zumindest eines *Gestaltungsrasters* für die Gestaltung der Oberflächen benutzen. Es stehen neben Rahmen für Fenstersysteme auch Rahmen für beliebig formatierbare Dokumente zur Verfügung. Ein solcher Rahmen wird in Analogie zu den üblichen Beziehungen

Anwendungsgebiet	Element	allgemeine Struktur
Datenbanksysteme	Tupel	Relationen-Typ
XML-Technologie	XML-Dokument	XSchema-Suite oder DTD-Suite
Benutzer-Schnittstellen-Technologie	Fenster	Stil-Regeln
XML-Generatoren	XSL-Regel	???
Kommunikationssysteme	???	???

entwickelt.

Diese Rahmen sind etwas komplexer als die Stil-Regeln für Benutzer-Schnittstellen, weil wir auch die Anwendergruppe, deren Profile und deren Portfolio mit berücksichtigen wollen. Zur Gestaltung entwickeln wir *Gestaltungsrahmen*, die die Art der Gestaltung, die allgemeinen Prinzipien und den Umgang mit multimedialen Elementen darstellen. Mit dem Gestaltungsrahmen wird vorgegeben, wie die Oberflächen gestaltet werden.

Außerdem sollen die Arbeitsoberflächen das Arbeiten mit dem System vereinfachen. Dazu erscheint es günstig, auch die Art des Zusammenwirkens, die Beziehungen der unterschiedlichen Akteure und die Darstellung des Zusammenwirkens durch den Arbeitsplatz zu kanonisieren. Dafür werden entsprechende *Kommunikationsrahmen* entwickelt. Die Art der Kollaboration bzw. Kooperation, die Art des Zusammenwirkens und der Arbeitsplatz werden mit berücksichtigt.

Wir berücksichtigen neben den bereits diskutierten Eigenschaften von Oberflächen die folgenden Gestaltungsmöglichkeiten.

Multimediale Darstellung: Einziger Zweck der Oberfläche ist es, etwas mitzuteilen. Sie ist niemals Selbstzweck, sondern steht im Dienste der Arbeit mit den Informationen. Durch die Einengung auf den Bildschirm wird die 'Vermittlung einer Botschaft' auch eingeschränkt. Eine Folge von Bildschirmen soll weder ermüden noch von der eigentlichen Arbeit ablenken.

Zugleich kann eine Oberfläche mehr Informationen vermitteln als ein einfaches Foto. Es werden Aktionen und Objekte in der Wechselwirkung sichtbar. Die 'Dekoration' ist jede Art von Hintergrund. Die Requisite kann entweder zur Dekoration oder zum Akteur gehören. Lichtwechsel und das Aussehen von Requisiten dienen der Gestaltung von Oberflächen.

Eine multimediale Arbeitsumgebung schließt die Verwendung von Tönen ein. Töne sind ebenso eine Informationsquelle. Die wichtigste Funktion des Tones liegt im Dialog mit dem

Akteur. Er ist die bei weitem einfachste Form der Faktenübermittlung. Er sollte jedoch nur dann angewandt werden, wenn andere Ausdrucksmöglichkeiten voll ausgeschöpft sind. Demgegenüber kann jede Aktion von bestimmten Geräuschen begleitet sein. Hintergrundmusik ist ein Bestandteil der Tonebene, jedoch i.a. nicht der Geschichte. Es können damit auch Informationen vermittelt werden.

Informationsquellen: Jede Oberfläche, jeder Dialogschritt vermittelt Informationen. Damit wird eine Oberfläche zur Informationsquelle. Die Vielfalt der Informationen kann auch durch die Kombination verstärkt, abgeschwächt oder auch beigeordnet werden. Durch eine neue Information kann auch eine Veränderung implizit angezeigt werden. Wird die Information komplexer, dann ist die Wiederholung ein nützliches und angebrachtes Mittel. Verdopplung kann verwendet werden, um Daten, die benötigt aber evt. vergessen werden, wieder in Erinnerung zu bringen. Typische Verdopplungsfunktionen sind Statusleisten. Sie sind jedoch mit Vorsicht zum richtigen Zeitpunkt mit der richtigen Information anzuwenden. Die Vielfalt an zu vermittelnder Information ist in geschickter Anordnung (Arrangement, Koordination) dem Akteur zu vermitteln. Dabei ist auch die semantische Konsistenz zu beachten. Widersprüche deuten auf Fehler hin.

Informationsquellen dürfen nicht mit Symbolismus verwechselt werden, der eher eine ungeeignete Art der Bildkonzeption ist.

Bildauswertung und Bildkomposition: Jede Szene in der Inszenierung und jede Oberfläche besteht aus kleinen Einheiten, die wiederum aus kleinen Einheiten zusammengefügt sein können. Sie setzen sich zu einer *Einstellung* zusammen, die sich auch je nach Betrachtungspunkt verändern kann. Das Blickfeld selbst ist begrenzt. Es wird nur ein relevanter bzw. wichtiger Ausschnitt gezeigt, d.h. die informationsträchtigen Elemente, die zur gleichen Aktion gehören, werden als Elemente einer Einstellung zusammengefaßt. Eine gute Einstellung hält mit den Aktionen und ihren Zielen Schritt. Mitunter ist die Reaktion wichtiger als die Aktion. Wir unterscheiden dabei verschiedene Einstellungstypen (Groß-, Nah-, halbtotale etc. Einstellung). Jede Einstellung kann auf unterschiedliche Weise informationsvermittelnden Fakten entsprechen. Eine Einstellung kann auch dynamisch sein und trägt damit u.a. der Verlagerung des Interesses Rechnung.

Ein Einstellungswechsel darf nicht zu kraß sein. Die Szenarien sollen durch eine nahtlose Verbindung von Einstellungen zusammenhängen. Mittel der Abgrenzung wie Auf- und Abblende sind deshalb in den Entwurf mit einzubinden.

Einzelszenen: Die Einzelszenen können auch aus mehreren Einstellungen bestehen. In diesem Fall sind auch mehrere Sichten auf die Datenbank zu integrieren. Ereignisse sind in den Szenen selbst enthalten. Einzelszenen sind durch ihren Ort, ihre Zeit (Zeitpunkt, Laufzeit, Länge) mitbestimmt. Eine kunstvolle Zusammenstellung von Elementen verbessert nicht unbedingt die Qualität der Inszenierung, es kann dadurch die Aufmerksamkeit der Arbeit entzogen werden. Damit wird die Exposition von Ort und Zeit zum Entwurfsproblem.

Auswahl der Informationen: Ein Szenario wird durch den Akteur als eine Folge von Einzelinformationen beobachtet. Alle Informationen beruhen in der Inszenierung auf Selektion. Es werden bestimmte Szenen herausgehoben oder auch nur angedeutet. Jede Information zieht auch weitere Informationen nach sich. Voraussetzung für die richtige Informationsauswahl ist daher die Kenntnis aller Fakten über den Dialogablauf und die zugrundegelegten Funktionen und Daten. Es können zu wenig, zu viel oder die korrekte Anzahl von wichtigen Fakten in den Entwurf eingehen. Dies trifft insbesondere auch auf die Darstellung der Begleitinformation zu. Zugleich werden die Vorkenntnisse der Benutzer mitberücksichtigt.

Der Informationspflicht am Anfang eines Szenarios muß in stärkerem Maß nachgekommen werden. Man kann auch Informationen, die später benötigt werden, vorher 'säen'.

Die ***Verteilung der Informationen*** unterliegt ebenso wie die Verteilung von Wissen und Nichtwissen komplizierten Gesetzmäßigkeiten und verstärkt die Wichtigkeit der Informationsvermittlung. Durch eine ungeschickte Verteilung können auch Mißverständnisse hervorgerufen werden.

Die Inszenierung bietet mit einer multimedialen Ausgestaltung des Drehbuches eine Vielfalt von Möglichkeiten, die, gerade weil sie existieren, danach verlangen, genutzt zu werden. Damit werden jedoch neue Hindernisse aufgetürmt, die die erfolgreiche Nutzung erschweren. Es ist nicht möglich, das gesamte Hintergrundwissen und den 'common-sense' in die Ausgestaltung zu integrieren. Obwohl wir viele Ereignisse präsentieren können, ist es schwierig, sie klar und verständlich zu präsentieren, weil i.a. keine beschreibenden und erklärenden Manual-Kurzgeschichten hinzugenommen werden sollten.

Abschließend bewerten wir die *Qualität* der Inszenierung.

Zieltechnik: Die Zieltechnik beeinflußt in sehr starkem Maße die Qualität und auch die Implementierbarkeit von einzelnen Szenen.

Einheitlichkeit: Neben Standardinteraktionen besitzen wir auch aus dem Inhalt abgeleitete Interaktionen. Eine Vereinheitlichung ist dabei angebracht.

Professionelles Design: Ein System soll einen Akteur nicht unterfordern, nicht überfluten und auch ein einfaches Wiedereinsteigen ermöglichen. Damit sind auch die Dialoge professionell zu gestalten.

Fehlerrobustheit: Eine Fehlbedienung darf weder zum 'core dump' noch zu unkontrollierbaren Zuständen führen. Ein Akteur muß selbst aus einer Fehlersituation wieder herausfinden.

Hierarchie der einzelnen Szenen: Da die Szenen geordnet werden, ist auch dem Akteur eine wiederholte Anwahl von einzelnen Szenen zu gestatten, so daß auch ein konsistentes, nach- und rückverfolgbares Szenenmanagement einen Benutzer unterstützen muß.

Farbauswahl: Wie jedes andere Darstellungsmittel sind auch die Farben mit einer semantischen Bedeutung zu versehen.

Darstellungsskalierung: Je nach Akteur, je nach vorhandenem Client oder Darstellungsmedium sind unterschiedliche Interaktionsmöglichkeiten vorzusehen.

Offene Systeme: Ein Informationssystem wird in immer stärkerem Maße mit anderen Systemen in integrierter Form verwendet. Deshalb ist der Output für einige Standards mit aufzubereiten.

Erweiterbarkeit: Ein Informationssystem beginnt aufgrund der Änderungen in der Anwendung selbst, in den Profilen der Akteure und durch Hinzunahme von Funktionalität bald nach der Erstellung 'zu leben'. Die möglichen Erweiterungen sollten antizipiert werden.

Auswahl der multimedialen Medien: Ein Akteur sollte entsprechend seinem Benutzerprofil die Interaktion und die benutzten multimedialen Formen selbst und evt. auch dynamisch auswählen können.

Benutzer- und Akteursmodelle

Wie bereits dargestellt, unterscheiden wir zwischen einem Benutzer und einem Akteur. Ein *Benutzer*[12] ist eine Repräsentation der aktuell agierenden Person z.B. durch die Login-Daten und die persönlichen Daten sowie die Benutzungsgeschichte. Benutzer werden im allgemeinen kategorisiert oder gruppiert.

Benutzer können nach ihren Eigenschaften gruppiert und mit einem Typkonzept dargestellt werden. Ein *Akteur* ist ein abstrakter Benutzer-Typ, der eine Gruppe von Benutzern abstrakt darstellt. Damit werden die allgemeinen Charakteristiken von Benutzern beschrieben. In der konzeptionellen Modellierung sind wir mehr an einer Darstellung von Akteuren orientiert.

Akteure sind den einzelnen Dialogschritten und damit den Szenen mit entsprechenden Rechten und Rollen zugeordnet. Diese Rollen erlauben einem Akteur das Agieren mit dem Informationssystem. Eine direkte Interaktion mit entsprechenden Funktionen über entsprechende Sichten oder das Schema direkt ist nach wie vor auch möglich. In diesem Fall wird jedoch nicht eine entsprechende Oberflächenmodellierung vorgenommen. Da solche Interaktionen in ihrer Vielfalt kaum zu behandeln sind, modellieren wir sie nicht gesondert, sondern benutzen die Dienste der logischen Schicht.

[12] [illegible]

Dieses Akteurmodell verallgemeinert das Use-Case-Modell. Wir streben eine möglichst abstrakte Beschreibung am Anfang an und gehen erst dann ins Detail, wenn der Endanwender nicht mehr involviert ist. Beziehungen zwischen den Akteuren werden nur durch entsprechende Dialoge aufgebaut. Die Beziehung zwischen Akteur und System findet hier jedoch nur durch entsprechende Dialoge statt. Ein Akteur aktiviert einen Dialog und erhält Daten aus dem Dialog, modifiziert Daten im Dialog oder stellt dem System Daten im Dialog zur Verfügung. Damit ist das hier angewandte Modell viel allgemeiner und zugleich praktikabler als das Use-Case-Modell.

Einem Akteur wird ein **Profil** zugeordnet. Es umfaßt

das Ausbildungsprofil mit einer allgemeinen Darstellung der erforderlichen Kenntnisse, Fähigkeiten und Fertigkeiten,

das Arbeitsprofil mit einer Darstellung der Spezifika der Akteure und in Ergänzung zum Ausbildungsprofil und

das Persönlichkeitsprofil zur Darstellung der Eigenschaften von Gruppen.

Das *Ausbildungsprofil* stellt für eine Gruppe von Benutzern das gesamte Spektrum der Ausbildung, die die Benutzer

- benötigen,
- mitbringen und ggf. auch
- nicht besitzen

sollen, dar. Die letzte Kategorie dient auch der Charakterisierung von *Wissens-, Fertigkeiten- und Fähigkeitslücken*. Diese Kategorie erlaubt auch eine Ableitung von Content, der für die Bewältigung der Arbeitsaufgabe durch das Informationssystem bereitgestellt werden muß.

Die erste Kategorie dient der Darstellung des *Bildungsweges*, der auch in grober Form dargestellt werden kann. Die Darstellung des Bildungsweges wird meist in analoger Form wie in Stellenanzeigen oder Stellenprofilen eines Arbeitsplatzes erfolgen. Wir benötigen diese Kategorie zur Ableitung der pragmatischen Annahmen, die eine Vereinfachung des Systemes bedingen.

Die zweite Kategorie kennzeichnet das *Bildungsspektrum* der Benutzer. Wird das Spektrum nicht berücksichtigt, dann verleitet ein System relativ schnell zum Mißbrauch oder wird abgelehnt, obwohl es gerade zur Bewältigung der Arbeitsaufgabe adäquat erscheint.

Das *Arbeitsprofil* ist analog zur KADS-Charakterisierung [LFe] auf eine Klassifikation der Akteure nach Eigenschaften ausgerichtet:

- Fähigkeiten, die Akteure zur Bewältigung der Arbeitsaufgaben besitzen sollen,
- Fertigkeiten, die zur Benutzung des Systemes erforderlich sind,
- Wissen, das zum Verständnis der Benutzung des Systemes erforderlich ist,
- Arbeitsumgebung, die durch die Akteure toleriert bzw. akzeptiert wird, und
- Systeme, mit denen die Akteure bereits Arbeitsaufgaben bewältigt haben.

Das Persönlichkeitsprofil umfaßt auch das *Polaritätenprofil*. Typische Polaritätenprofile sind nach Anmutungscharakteren sachlich-romantisch, konventionell-originell, klassisch-modisch, traditionell-avantgardistisch, tough-tender, rustikal-artifiziell und einfach-wertvoll. Im Einzelnen können wir dazu Differenzierungen nach Notenskalen für die Antonyme vornehmen.

sachlich	-	romantisch	konventionell	-	originell
nüchtern	-	gefühlvoll	üblich	-	ausgeflippt
rational	-	sensitiv	bedeckt	-	frisch
überlegt	-	sinnlich	seriös	-	ungewöhnlich
			bürgerlich	-	bohemehaft
klassisch	-	modisch	traditionell	-	avantgardistisch
dezent	-	laut	alt	-	jung
zeitlos	-	modern	uni	-	bunt
ruhig	-	unruhig	ruhig	-	erregend
zurückhaltend	-	aufdringlich	vertraut	-	vertraut
			gewohnt	-	poppig
tough	-	tender	rustikal	-	artifiziell
herb	-	süßlich	natürlich	-	künstlich
geometrisch	-	blumig	verspielt	-	streng
hart	-	weich	einfach	-	komplex
robust	-	zart	schwer	-	leicht
eckig	-	rund	grob	-	grazil

Daraus kann die Charakterisierung von Benutzergruppen abgeleitet werden.

Bekannt ist z.B. nach [KT95] das Fremdbild wie in Bild 40.

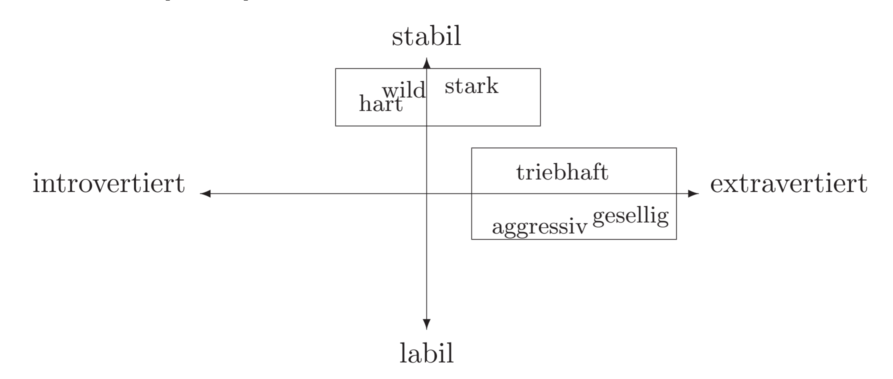


Bild 40: Das Fremdbild des Bayern

Ähnliche Profile sind auch für andere Gruppen bekannt. Mit diesen Profilen können Portfolios für die einzelnen Benutzergruppen erstellt werden. Hinzu kommen dabei auch noch Morphologien. Ein Oberflächen-Portfolio setzt sich dabei aus mehreren ebenen Profilen zusammen wie Funktion-Semantik, Prägnanz-Expressivität.

Zum Persönlichkeitsprofil gehört auch das subjektive *Informationsbedürfnis*, das wiederum abhängig von der (intuitiven) Erkenntnis ist, daß die vorhandene Information zur Lösung einer Aufgabe nicht ausreicht, daß das Wissen zu gering ist, daß die Information aus vorhandenen Wissen und Informationen nicht oder nicht so schnell generiert werden kann, daß die Unsicherheit, Unbestimmtheit, Unschärfe oder die Widersprüche nicht anders aufgelöst werden können. Wir unterscheiden den Informationsbedarf vom Informationsbedürfnis. Das *Informationsbedürfnis* ist abstrakt ein Wunsch nach besserer Information. Der Informationsbedarf wird für das Portfolio benötigt.

Bei der Entwicklung von Informationssystemen sind unterschiedliche Informationsbedürfnisse entsprechend dem Profil zu beachten. Damit kann ein Informationssystem nur dann von Bestand sein, wenn es ein Bündel von Informationsdiensten aus den folgenden Kategorien bereitstellt.

Informationsdienste im allgemeinen Interesse orientieren sich insbesondere analog zu Zeitungen auf die Bereitstellung von Informationen des täglichen Alltags. Beispiele sind Wetterdienste, Veranstaltungskalender, Regionalinformationen, Sportinformationen und Nachrichtendienste.

Informationsdienste zur Gestaltung der Freizeit orientieren sich z.Z. noch am Computerspielmarkt, werden aber auch immer stärker Aufgaben der Kommunikation (wie auch Email) übernehmen und sich zunehmend in eine stärkere Konkurrenz mit Printmedien wie Nachschlagewerke, Verzeichnisse wie Adreßbücher begeben.

Informationsdienste zur Erfüllung von Arbeitsaufgaben werden zuerst als allgemeine Betriebsinformationssysteme (wie z.B. als Campusinformationssystem) erfolgreich sein. Die Achillesferse der

heute vorrangig entwickelten Wirtschaftsinformationsdienste ist die Aktualität der angebotenen Information[13].

Jede Gruppe von Benutzern besitzt auch Spezifika. Diese ergänzen das allgemeine Profil um folgende Informationen:

positive Arbeitserfahrungen für die Anwendung wie *praktizierte Kenntnisse*, *Lösungsstrategien* und *Fertigkeiten bei der Anwendung* des eigenen Wissens,

negative Arbeitserfahrungen (i.a. *Fehlersuche, Fehlerbeseitigung*, *Arbeitsplanung*, *Arbeitsschrittentscheidungen*, *Bewertung* des Arbeitsfortschrittes, *Konstruktion der Lösungen*, Umgang mit *Abstraktionstechniken*, *Effektivität*, *Erweiterung* für bereits gefundene Teillösungen und *Kooperationsfähigkeit*) und

spezielle Kenntnisse (wie *(Wissens-)Repäsentationstechniken*, *(Wissens-)Akquisition* und andere).

Die Profile von Akteuren können kategorisiert und damit einer Skalierung unterzogen werden. Wir können z.B. mit der folgenden Kategorisierung die Profile der Akteure zum *Erstellen eines Lehrveranstaltungsvorschlages eines Lehrstuhles* vornehmen:

Ausbildungsprofil			Arbeitsprofil					Persönlichkeitsprofil		Folgerung für Umgebung
erforderlich	vorhanden	nicht vorh.	Fähigkeiten	Fertigkeiten	Wissen	Arbeitsumgebung	System	Polaritätenprofil	...	
Informatik	*Informatik*	*Organisationserfahrung*	*Java, C++*	*Programmierung*	*Informatik*	*Unix*	*Workstation*	*rigide*	...	*Kommandosprache, ohne Sicherung*
Büro-Kauffrau/ -mann	*PH-Studium*	*Informatik*	*Organisator*	*kollaborativ*	*allg.*	*PC-Platz*	*minimal*	*moderat*	...	*Fehlertoleranz, Übersichtlichkeit*
...	...	...	...	...	...	...	...	...	...	...

Andere ableitbare Eigenschaften sind z.B. die erforderlich Hilfe, die Art des Systemerlernens, die Adaptivität der Interfaces, die Erweiterbarkeit, exploratives Handeln, selbst gesteuerte Nutzung, Merkhilfen, Aufgabenorientierung, Routinetoleranz, Technikerwartungen, Zusatzaufwandtoleranz, EDV-Terminologie-Toleranz, Aufgabenbezug, Benutzerführung, Beispiele, Einführung und Voreinstellungen.

Aus dem Profil können wir die Art und die Form der *Informationspräsentation* und das *Informationsbeschaffungsverhalten* der Akteure ableiten. Weiterhin kann man *Benutzungspräferenzen* der Akteure skizzieren.

Akteure können mit anderen Akteuren zusammenwirken. Im Zusammenwirken spielen Ziele eine Rolle. Ein Modell zur Darstellung der Ziele stellen wir in Bild 41 kurz zusammen.

Im Zielmodell unterscheiden wir zwischen unscharfen oder "weichen" Zielen und "harten" Zielen. Weiche Ziele besitzen kein genau darstellbares Erfüllungskriterium. Harte Ziele sind dagegen durch ein Erfüllungskriterium charakterisiert. Zum Erreichen von Zielen können Akteure zusammenarbeiten.

Einem Akteur kann ein *Sicherheitsprofil* zugeordnet werden. Wir verwenden dazu eine Datenstruktur wie in Bild 42.

[13]Die Erfahrung im Projekt FuE*line* deutete auf eine Halbwertszeit von weniger als 3 Monaten hin, wodurch der Verfall eines wie perfekt auch immer gestalteten Informationsbestandes innerhalb kurzer Zeit vollständig ist, wenn nicht

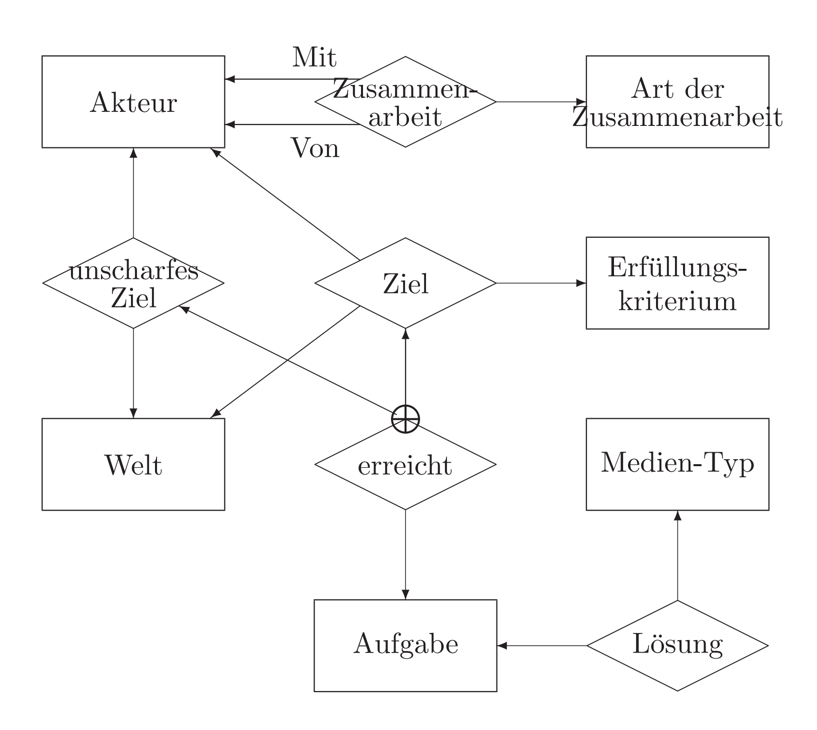


Bild 41: Die Zusammenarbeit von Akteuren zum Erreichen von Zielen

Das Sicherheitsprofil eines Akteurs wird durch Sicherheitsoptionen, mit denen die gesamte Sicherung des Systemes dargestellt werden kann, charakterisiert. Zur Durchführung von Aufgaben im Rahmen des Story-Raumes werden entsprechende Medien-Typen bereitgestellt. Da diese den Aufgaben zugeordnet sind, werden Sicherheitsoptionen mit vier Parametern spezifiziert.

Akteure werden mit entsprechenden Sicherheitsoptionen zur Erfüllung von Aufgaben ausgestattet.

Aufgaben determinieren die erlaubten Aktionen, erfordern Aktionen oder determinieren spezifische Sicherheitsprofile.

Rollen von Akteuren entsprechen den bereits besprochenen Rollen im Story-Raum. Für Sicherheitsprofile sind außerdem Rollen von Interesse, die einer Gruppe von Akteuren zugeordnet werden.

Eine Sicherheitsoption basiert entweder auf erlaubten Aktionen oder expliziten Verboten.

Ein Benutzer wird einem Akteur zugeordnet. Er kann gleichzeitig einer Reihe von Akteuren zugeordnet werden.

Die Darstellung des Arbeitsrahmens und Benutzer- und Akteurportfolio

Das **Portfolio** des Akteurs wird beschrieben durch:

- die Beschreibung des Inhaltes der Aufgabe,
- die Spezifikation der Rechte des Akteurs im entsprechenden Dialogschritt,
- die Beschreibung der Rolle des Akteurs und
- die Ausführungsmodelle für das Agieren mit Angaben zur Zeitdauer (minimal, maximal, normal), sowie zu den Ausführungsprioritäten.

Eine *Aufgabe* ist eine Vorgabe für zielorientiertes Handeln und wird durch die folgenden Aspekte beschrieben:

- Die Darstellung der Aufgaben geht von einer Zielstruktur aus. Diese Zielstruktur kann im zustandsorientierten Zugang zur Modellierung durch Angabe des Zielzustandes erfaßt werden.

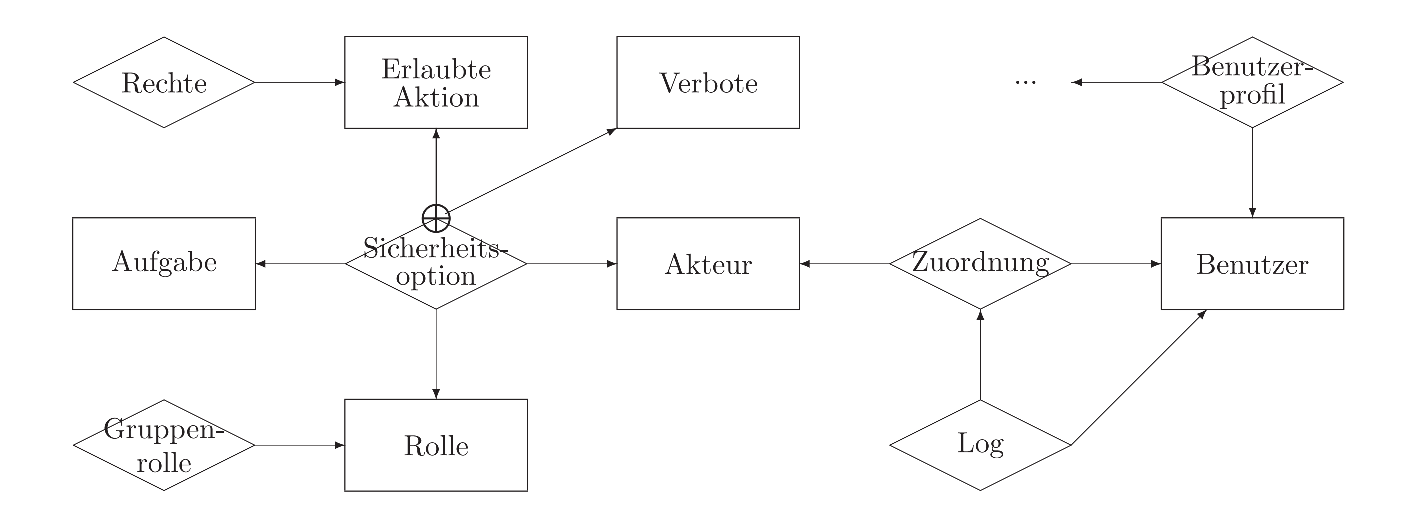


Bild 42: Das Sicherheitsprofil von Akteuren

- Durch eine Wissensprofil werden die Details des Aufgabenwissens erfaßt.
- Die Beschreibung der Arbeitsmittel basiert auf der Darstellung des Content und der erforderlichen Funktionalität.
- Die Erfüllung einer Aufgabe erfolgt in Arbeitsabläufen, die in einzelne Arbeitsvorgänge strukturiert sind.
- Es kann ein allgemeines Abarbeitungsmodell für die Wege zum Zielzustand vorgegeben sein. In stark strukturierten Arbeitsfeldern wird gerade auf die genaue und detaillierte Darstellung dieses Abarbeitungsmodelles viel Wert gelegt.

Eine Spezifikation der *Arbeitsvorgänge* umfaßt folgende Bestandteile:

Die allgemeine Struktur wird beschrieben durch

- einen Auslöser,
- eine organisatorische Einheit,
- eine Tätigkeit des Benutzers,
- verwendete Hilfsmittel und
- eine Ablage und einen Adressaten.

Das Resultat der Ausführung führt zu einem

- einem Ergebnis,
- das unter bestimmten Bedingungen akzeptiert wird.

Die semantischen Rahmenbedingungen sind definiert durch

- Bedingungen, unter denen der Arbeitsvorgang ausgeführt werden kann, und
- organisatorische Randbedingungen.

Arbeitsabläufe werden durch *Aktivitätenfolgendiagramme* beschrieben. Sie bestehen aus

einem Aktivitätstyp zur Kategorisierung von gleichartigen Aktivitäten,

einer Transition von Input/Outputdaten durch die Aktivität und

der Steuerung des Beginns, der Verzweigung etc. und der Beendigung einer Aktivität.

Mit dieser Spezifikation können wir Aktivitätenfolgendiagramme mit Workflow-Programmen assoziieren. Aktivitätenfolgendiagramme können sowohl zustandsorientiert durch Zustandsaktivitätendiagramme als auch ereignisorientiert durch Ereignisfolgendiagramme dargestellt werden.

Rechte eines Akteurs werden durch Zuordnung von Funktionen des Content-Objekt-Suite dargestellt, die ein Akteur zur Erfüllung der Arbeitsaufgabe erhält. Mit der expliziten Zuordnung wird ggf. der Spezifikationsaufwand höher. Wir können jedoch diese Zuordnung auch durch entsprechende Rechtetypen darstellen. Damit wird für die Spezifikation der Rechte nur eine Zuordnung zum Typ erforderlich.

Die Rolle eines Akteurs baut auf einer *Kategorisierung der Erfüllung* der Arbeitsaufgabe und auf dem *Organisationsmodell* auf.

Das Ausführungsmodell besteht aus

einem Aufgabenaufruf, mit dem die Ausführung initiiert werden kann,

einem Datenaustausch, mit der benötigte Daten für die Ausführung bereitgestellt und wieder in das Informationssystem eingepflegt werden können,

einer Aufgabenablaufsteuerung, mit der sequentielle und nebenläufige Abläufe dargestellt werden

einem Arbeitsbereich, auf dem mehrere unterschiedliche Aufgaben abgelegt werden können, und

einem Synchronisationsmodell, zum Abgleich der Ausführung von Aufgaben, die sich im Arbeitsbereich befinden.

5ergänzt werden.

Aus der Darstellung der Aufgaben können wir den *Informationsbedarf* ableiten. Der Informationsbedarf ist nach einer genauen Analyse des augenblicklichen Wissensstandes und der Ziele der Wissensvermittlung ableitbar und sogar in Geschäftsprozesse abbildbar. Die Qualität der Aufbereitung von Informationen wird durch den augenblicklichen Informationsbedarf mit determiniert.

Das Portfolio wird mit den Arbeitsgestaltungsdimensionen für die Gestaltung humaner Arbeit erweitert:

Der Entscheidungsspielraum kennzeichnet das Ausmaß, in dem ein Benutzer seinen Arbeitsprozeß selbst gestalten kann.

Die arbeitsbezogene Kollaboration dient der Abstimmung von Teilen der Arbeitsaufgabe mit anderen Akteuren.

Einschränkungen durch psychische Belastungen können durch entsprechende Hilfsmittel minimiert werden.

Der Zeitrahmen kennzeichnet die Möglichkeit, den Arbeitsablauf zeitlich selbständig durch den Akteur zu gestalten.

Die Variabilität ist bestimmt durch den Zusammenhang der Arbeitsvorgänge und der Vorgehensweise zur Aufgabenerfüllung.

Die Wahrnehmungen des Benutzers unterstützen die schnellere Erfassung der anstehenden Aufgaben.

Die körperliche Aktivität unterstützt die Erfüllung der Aufgaben.

Die Strukturierbarkeit des Arbeitsbereiches erlaubt eine Anpassung an die Arbeitsweise und Arbeitsmethodik des Benutzers.

Zur Spezifikation der Arbeitsgestaltungsdimension verwenden wir ein *Gestaltungspolaritätenprofil* mit entsprechenden antonymen Charakterisierungen wie z.B. für den Arbeitsvorgang zum *Erstellen der Vorschläge eines Lehrstuhles für Lehrveranstaltungen*:

Spielraum	Kollaboration	Belastung	Zeitrahmen	Variabilität	Wahrnehmung	Aktivität	Strukturierbarkeit
vollkommen eigenständig	*Abstimmung im Lehrstuhl*	*nebenläufige Tätigkeit*	*Ablieferungszeitpunkt*	*hoch, mit Sitzung, volle Unterstützung*	*wohlstrukturiert, ohne Maus*	*Direkteingabe, Übernahme vorhandener Daten*	*Aufgabenblatt mit Ordnung nach Erfüllungsstand*

Durch Interaktionsdiagramme werden die Story, die Szenario und das Drehbuch unterlegt. Interaktionsdiagramme sind gerichtete Graphen, deren Knoten Zustände des Systemes und deren Kanten Transitionen darstellen, die durch einen Akteur ausgelöst werden können. Es kann ein Akteur in seinen Rollen, mit seinen Rechten bei der Aufgabenlösung dargestellt werden. Das Akteurmodel faßt folgende Eigenschaften zusammen:

Profil des Akteurs,

Arbeitsziele des Akteurs,

Sicherheitsprofil des Akteurs und

Portfolio des Akteurs.

Wir trennen davon jedoch im Gegensatz zu Use-Case die Beziehungen der Dialoge zu den Daten und zu den Funktionen. Diese Trennung entspricht der klassischen Vorgehensweise und verhindert ein Überladen der Konstrukte. Damit sind außerdem auch die dort geforderten Ressourcenmodelle, Organigramme, Firmenstrategiemodelle, etc. nicht mehr notwendig. Mit der Verbindung zu den Sichten erhalten wir eine *Seiteninhaltsbeschreibung*.

Der Story-Raum, Szenen, Dialogschritte und Szenario

Wir unterscheiden zwischen dem Teil eines Systemes, der für den Benutzer sichtbar ist, und dem internen Teil eines Systemes, der für den Benutzer nicht sichtbar gemacht werden muß. Nach Wegner's Theorie der interaktiven Maschinen werden damit unterschiedliche Aspekte erfaßt. Interaktive Maschinen stellen die Interaktion eines Benutzers dar. Sie unterliegen anderen Paradigmen als klassische Berechnungssysteme:

Input-Output-Ströme: Ein Benutzer reagiert auf den Output eines Systemes mit seiner Antwort.

Beobachtungen, Glauben, Bedürfnisse, Intentionen und Aufgaben eines Akteurs bestimmen die Interpretation des beobachteten Output des Systemes mit.

- Damit besitzt die Antwort des Akteurs auf den beobachteten Output eine intensionale Logik.
- Ein Akteur stellt die Beobachtung zu den Aufgaben in Beziehung, die er gerade lösen möchte.
- Der Output wird mit einer Umgebung bzw. einen Kontext im allgemeinen in Beziehung gesetzt.

Damit wird durch den Akteur eine andere Beziehung eingegangen als dies bei der Modellierung von algorithmischen Systemen üblich ist. Mensch-Maschinen-Schnittstellen erfordern deshalb eine explizite Berücksichtigung folgender Parameter:

Beobachtetes Verhalten: Die Beobachtungen bestimmen die Sicht des Akteurs auf das System.

Interpretiertes Verhalten: Ein Akteur interpretiert das Verhalten des Systemes.

Nichtalgorithmisches Verhalten des Akteurs: Das Verhalten eines Akteurs ist meist nicht algorithmisch beschreibbar.

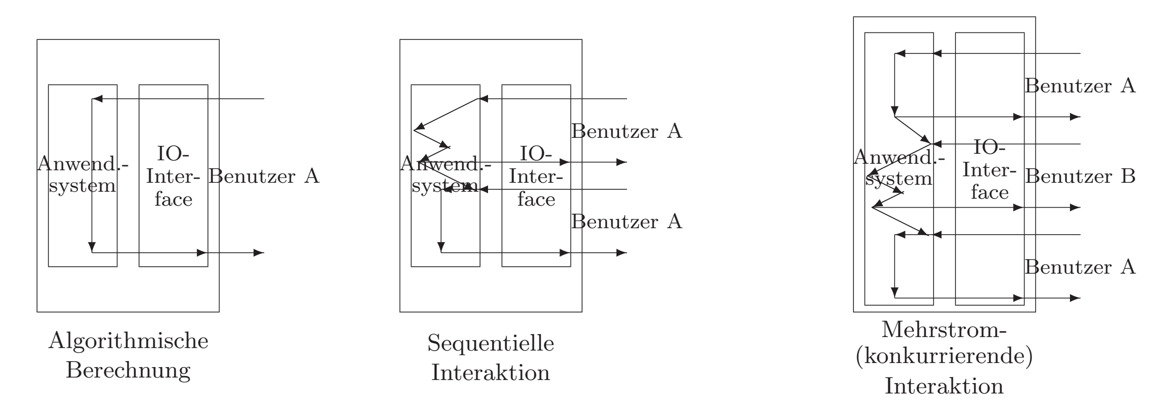


Bild 43: Algorithmisches Verhalten versus Mensch-Maschine-Verhaltes eines oder mehrerer Akteure

In Bild 43 vergleichen wir das algorithmische Verhalten eines Systemes in der klassischen Vorstellung mit der sequentiellen Interaktion, bei der auch das System ohne Benutzerinteraktion seinen Zustand ändert, wobei diese Änderung ggf. auch für einen Benutzer nicht mehr verstehbar oder nachvollziehbar ist. Das Verhalten wird noch weniger einsichtig, wenn das System seinen Zustand aufgrund einer Interaktion mehrerer Benutzer ändert. In letzteren Fall kann dadurch das Verhalten eines Systemes für den Einzelbenutzer zufällig oder chaotisch aussehen, obwohl das System selbst deterministisch ist.

Wir unterscheiden deshalb in Bild 44 zwischen

dem Systemraum, der das Systemverhalten widerspiegelt und für den wir das erweiterte ER-Modell zur Spezifikation verwenden, und

dem Interaktionsraum, der den Content des Benutzers enthält, auf einer Begriffs-, Konzept- oder Content-Algebra aufsetzt, aber einer anderen Logik unterliegt.

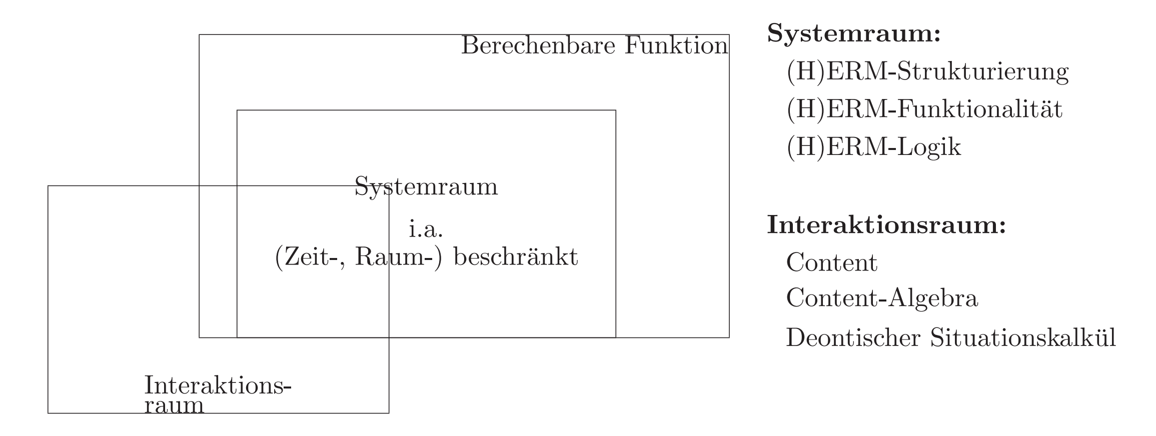


Bild 44: Der Interaktionsraum verglichen mit dem Systemraum

Der Interaktionsraum setzt in unserem Verständnis auf dem Systemraum auf, erweitert diesen jedoch um Benutzeraspekte.

Wir fassen diesen Spezifikationsansatz in der Sprache **SiteLang** zur Entwicklung von Storyboards zusammen. Wir führen dazu weitere Begriffe ein:

Der Story-Raum widerspiegelt die Menge aller Stories. Der Story-Raum besteht aus Szenen und markierten Transitionen zwischen diesen Szenen.

Eine Story stellt einen Handlungsstrom mit allen seinen Varianten dar.

Ein Szenarium ist ein Durchlauf durch eine Story, d.h. ein “Objekt” einer Story, wenn wir die Story als Klasse auffassen.

Szenario stellen ein Bündel oder einen Pfad von Durchläufen dar. Szenario können zu einer Story zusammengefaßt werden. Innerhalb eines Story-Raumes können viele Stories realisiert werden. Neben den Stories können auch Nebenstories und Hauptstories spezifiziert werden.

Eine Story besteht aus Szenen, in denen Akteuren ihre Content-Suite in ihrem Repräsentationstil und in Abhängigkeit von ihrem Kontext dargestellt wird.

Der Akteur stellt eine Gruppe von Benutzern in abstrakter Form dar.

Eine Szene besteht aus Dialogschritten, in denen die zugelassenen Akteure bestimmte Aktionen unternehmen. Die Markierung von Szenen wird beschrieben durch Ereignisse oder Aktivitäten für den Übergang von einer Szene zur nächsten, durch die involvierten Akteuere, durch Vor- und Nachbedingungen für die Nutzung der Szene, durch die Priorität der Transition, durch die Häufigkeit und durch die Anzahl der Wiederholungen.

Dialogschritte sind beschrieben durch die Sichten auf die Content-Objekte, die Manipulationsanforderungen, die zugelassenen Operationen, die Vorbedingung, eine Abschlußbedingung und die Ereignisse, die zum Schritt führen, sowie die agierenden Akteure der Szene.

Formal können wir diese Begriffe von SiteLang wie folgt einführen:

Der Story-Raum ist ein gerichteter, bewerteter Graph bestehend aus Szenen und den Transitionen zwischen den Szenen, d.h.

$$\begin{aligned}
&\text{Story-Raum} = (\{\text{ Szene }\}, E, \lambda, \kappa)\\
&\qquad E \subseteq \{\text{ Szene }\} \times \{\text{ Szene }\}\\
&\qquad \lambda : \{\text{ Szene }\} \rightarrow \text{SzeneBeschreibung}\\
&\qquad \kappa : E \rightarrow \text{TransitionBeschreibung}\\
&\text{TransitionBeschreibung} \subseteq\\
&\qquad (\text{Ereignisse} \cup \text{Aktivitäten}) \times \text{Akteure} \times \text{Vorbedingung} \times \text{Nachbedingung} \times\\
&\qquad\qquad \text{Priorität} \times \text{Häufigkeit} \times \text{Wiederholrate}
\end{aligned}$$

Eine Szene besteht aus Dialogausdrücken, dem Content, einer Darstellung der Akteure, einer Repräsentation und dem Kontext, d.h.

Szene = (SzeneID ,
  DialogueSchrittAusdruck ,
  Content-Suite,
  Akteure (
    AkteurID,
    Rechte,
    Aufgaben(
      Zuordnung,
      Rolle)),
  Repräsentation (Stil, Defaultwerte, Betonung, ...),
  Kontext (Hardware, Software, Kanal, Intention))

Datenbank-Unterstützung für den Story-Raum

Es sind verschiedene Repräsentationen für Szenen und Dialogschritte möglich. Für komplexere Anwendungen ist eine Datenbankablage der Elemente von SiteLang wünschenswert. Dies kann durch eine Struktur wie in Bild 45 erfolgen. Damit sind dann auch in einfacher Form einzelne Schritte eines Szenario abwandelbar, ohne im Detail alle Strukturen, Oberflächen und Prozesse durchmustern zu müssen.

Der Vorteil dieser Abbildung des Story-Raumes liegt auf der Hand: Es können Änderungen jederzeit in einfacher Form eingebracht werden, ohne sich direkt auf die gesamte Prozeßwelt auszuwirken.

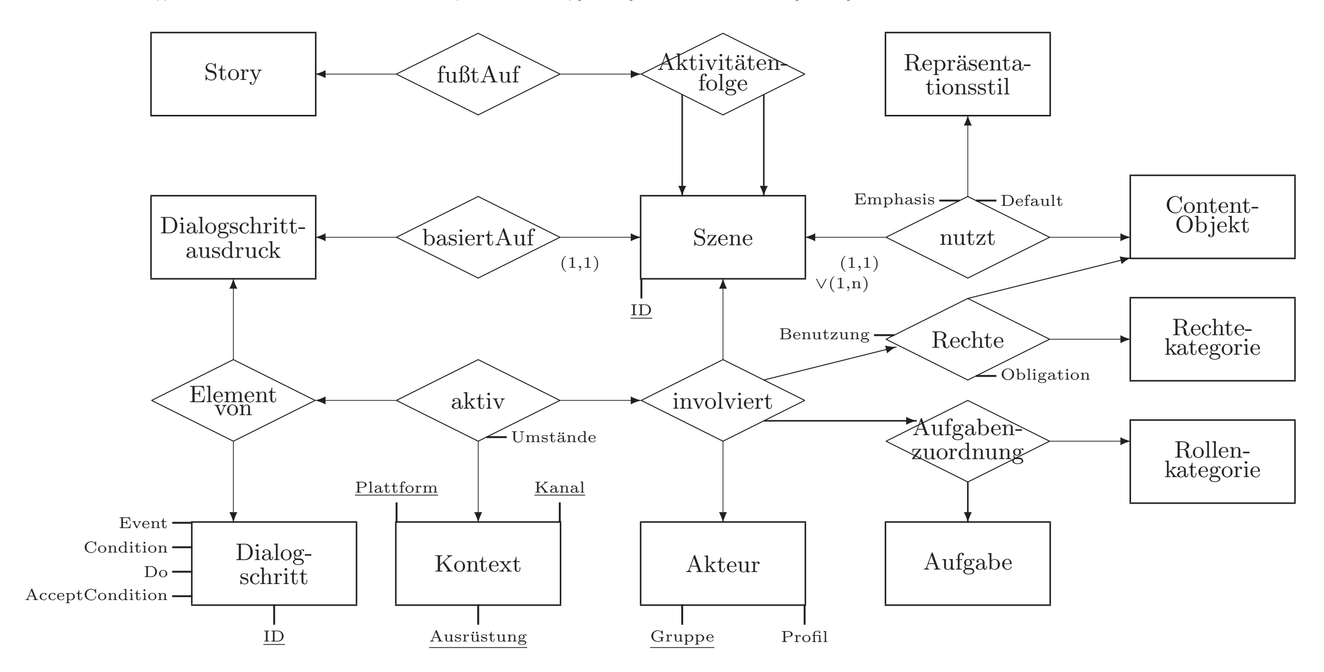


Bild 45: Repräsentation der Elemente von SiteLang

Das Ausspiel der Oberfläche wird durch entsprechende XML-Architekturen besonders unterstützt. In diesem Fall kann mit einer Architektur nach dem Zwiebelprinzip in Bild 46 vorgegangen werden. Mit dieser Generierung erreicht man nicht nur eine höhere Flexibilität, sondern auch eine Vereinfachung der gesamten Anwendung.

Direktdarstellung des Story-Raumes

Eine Datenbank-Unterstützung ist nicht in jedem Fall für den Story-Raum notwendig. Wir können auch anstelle einer vollständigen Datenbank direkt die folgenden OLAP-Elemente betrachtet werden, die z.T. allerdings redundante Informationen enthalten:

Dialogszene: In einer Dialogszene werden die involvierten Akteure, das genutzte Content-Objekt und die Dialogschritte beschrieben.

Dialogschritt: Ein Dialogschritt ist die kleinste Story-Einheit. Sie erlaubt die Darstellung der Retrieval-Sichten, der bereitgestellten Funktionen, einer Einschränkung der involvierten Akteure auf aktive Akteure und einer Steuerspezifikation mit

- Ereignissen, die zum Aufsuchen dieses Dialogschritts führen,
- Bedingungen, unter denen der Dialogschritt ausgeführt werden kann, und
- Beendigungsbedingungen, mit denen eine explizite Kontrolle realisiert werden kann, so daß ein Dialogschritt erst beendet werden kann, wenn eine bestimmte Konsistenz erreicht ist.

Szenario: Der Story-Raum erlaubt eine Vielzahl von Durchläufen. Da in einer Anwendung nur einige Durchläufe von Interesse sind, spezifizieren wir die Hauptdurchläufe durch Szenario. Szenario sind i.a adaptierbar an die Benutzung, an die direkten Benutzer, deren Anwendungskontext und deren Benutzungshistorie. Dies wird durch eine Parametrisierung erreicht.

Szenarium: Jedes Szenario enthält aufgrund der Parametrisierung eine Vielzahl von Durchläufen. Ein konkreter Durchlauf wird durch eine Wertezuweisung an alle Parameter zu einem Szenarium.

Diese Direktspezifikation wird insbesondere für Informationssysteme angewandt, deren Präsentati-

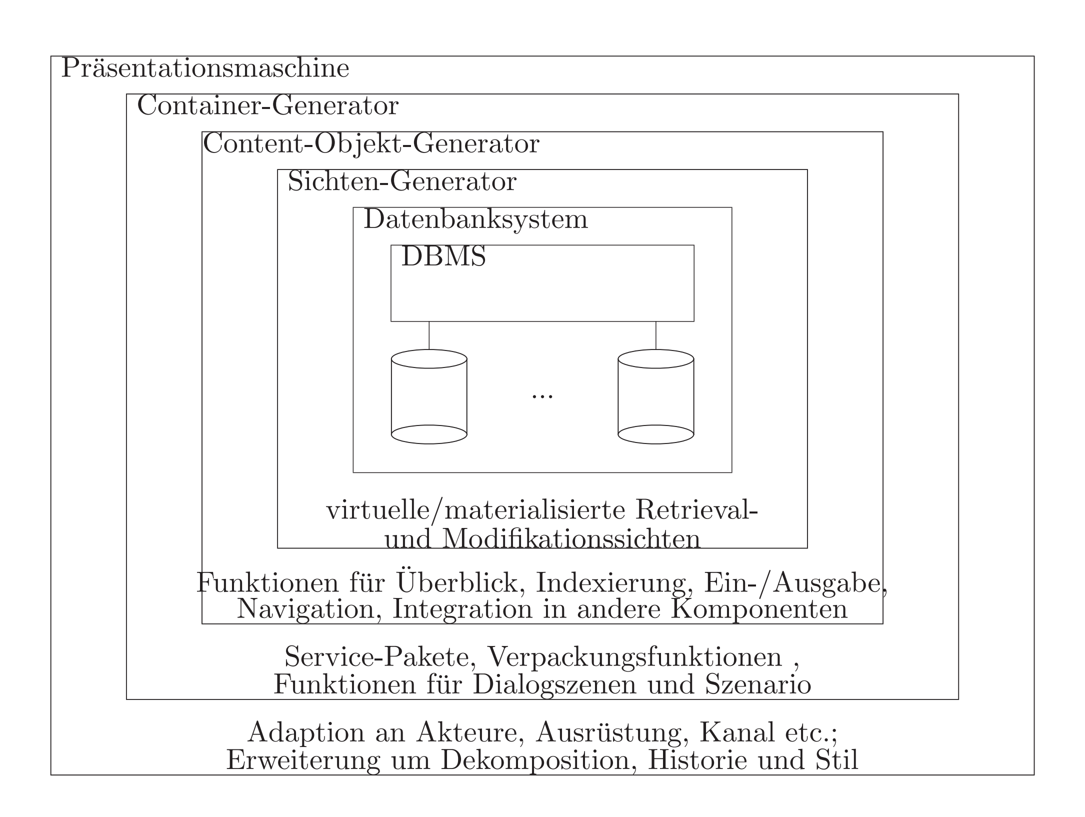


Bild 46: Der Zwiebelzugang zur Generierung der Oberflächen von Anwendungen

Einführung von Identifikatoren für die Elemente sinnvoll.

Dialogschritte können spezifiziert werden durch Tabellen der folgenden Form:

Dialog-schritt-name	on event	Vorbedingung	Content-Objekt	zugelassene Operationen	zugelassene Manipulations-operationen	Akteur	accept on
....	....	...	....	....	....	....	....

Es sind auch graphische Repräsentationen wie in Bild 47 sinnvoll.

Der Spezifikationsrahmen für Dialogschritte

Die Spezifikation der einzelnen Dialogschritte wird in sechs Dimensionen aufgefächert:

Die Intentionen der einzelnen Dialogschritte folgen der allgemeinen Mission der Anwendung und werden durch entsprechende Metaphorik gut unterstützt.

Der Story-Raum wird durch die Handlungsverläufe der Anwendung bestimmt. Er zerfällt in Szenen, die wiederum in Dialogschritte zerlegt werden.

Die Spezifikation der Benutzung basiert auf einer Darstellung der Akteure, ihrer Rollen, Rechte und Verantwortlichkeiten, sowie der Präsentation.

Der Content der einzelnen Dialogschritte wird durch eine Content-Objekt-Suite bestimmt.

Die unterstützende Funktionalität für die einzelnen Dialogschritte wird auf der Funktionalität der Content-Typen aufgesetzt.

Der Kontext der einzelnen Dialogschritte wird durch den Kontextraum determiniert.

Diese sechs Dimensionen können in Zusammenhag mit dem Zachman-Spezifikationsrahmen gestellt werden. Wir unterschieden vier Hauptdimensionen für jeden Dialogschritt:

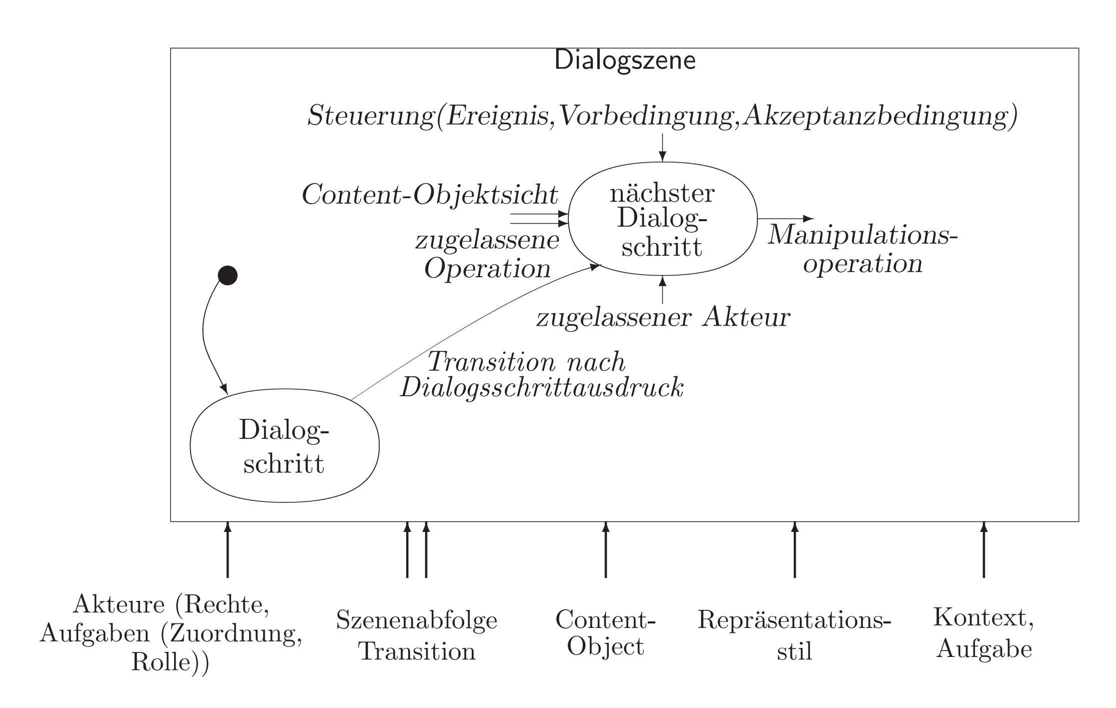


Bild 47: Sichtenstern für eine Dialogszene mit entsprechenden SiteLang-Elementen

die zugelassenen ***Akteure*** des Dialogschrittes,

die Einbettung in den ***Story-Raum,***

die bereitgestellten ***Content-Objekte*** und

der ***Zeitrahmen*** für die Absolvierung des Dialogschrittes.

Diese Hauptdimensionen sind in Bild 48 graphisch mit ihren Assoziationen skizziert.

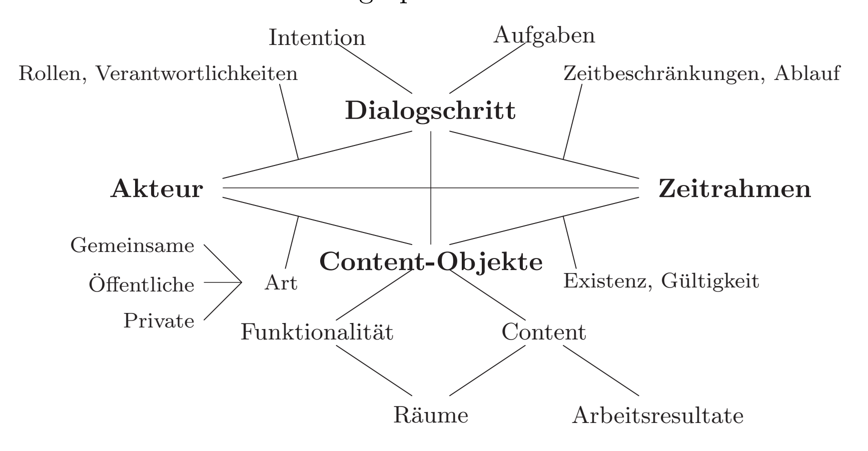


Bild 48: Der Spezifikationsrahmen für Dialogschritte

Die Assoziationen werden gleichzeitig mit dem Rahmen dargestellt. So sind z.B. Content-Objekte mit Akteuren assoziiert. Sie können öffentlich sein, von Akteuren gemeinsam benutzt werden oder auch privat sein. Akteure beteiligen sich in Dialogschritten in unterschiedlichen Rollen und Verantwortlichkeiten. Die Content-Objekte werden als Content-Suite mit einer entsprechenden Funktionalität bereitgestellt. Für Arbeitsgruppen sind Dokumente und Arbeitsräume typische Content-Objekte. Die Content-Objekte sind ggf. nicht dauerhaft gültig und können erzeugt, modifiziert und gelöscht werden. Die Dialogschritte werden zur Bewältigung von Aufgaben mit bestimmten Intentionen benutzt.

Als typisches Beispiel kann die Untersetzung des Spezifikationsrahmens für Dialogschritte von Arbeitsgruppen-Sites wie in Bild 49 angegeben werden:

Die ***Akteure*** sind Arbeitsgruppenmitglieder, Arbeitsgruppenleiter und -verantwortliche, insbesondere Administratoren.

Der ***Story-Raum*** besteht aus einer Reihe von Dialogschritten wie z.B. dem Zusammenarbeitsschritt.

Die ***Content-Objekte,*** z.B. die Gruppendokumente, können auch spezielle Dokumente wie Tagesordnungen, allgemeine Nachrichten oder persönliche Mitteilungen sein. Öffentliche Dokumente werden in Wandzeitungen etc. veröffentlicht. Dokumente können verpackt und entpackt, editiert oder auch nur gelesen werden. Es werden den Mitgliedern eigene Arbeitsräume, z.B. Schreibtische und persönliche Speicher, zur Verfügung gestellt.

Der ***Zeitrahmen*** wird durch die Zusammenarbeit der Arbeitsgruppe vorgegeben.

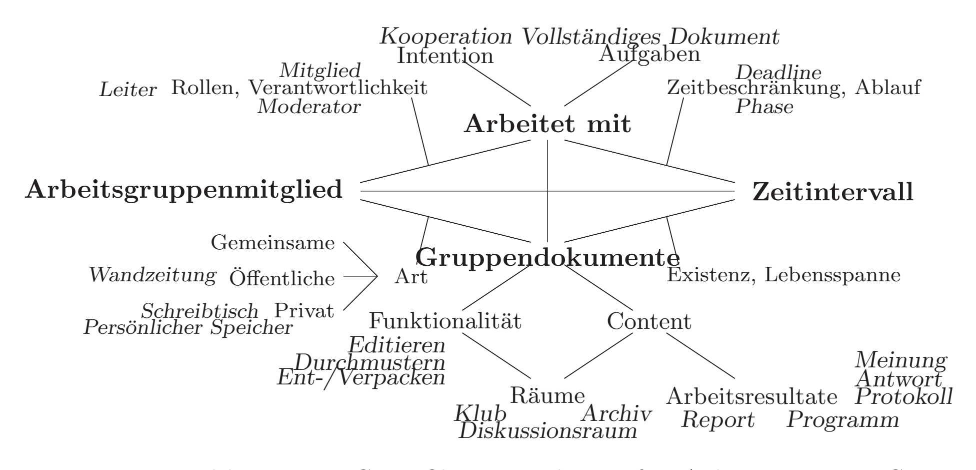


Bild 49: Der Spezifikationsrahmen für Arbeitsgruppen-Sites

Der Spezifikationsrahmen für einen Beitrag eines Arbeitsgruppenmitgliedes wird in Bild 50 vorgestellt:

Die ***Akteure*** sind diesmal Mitglieder einer Redaktionskommission. Sie erstellen, kommentieren und lesen gemeinsame Beiträge.

Der ***Story-Raum*** umfaßt z.B. den Dialogschritt einer Beitragserstellung.

Die ***Content-Objekte*** sind Beiträge. Sie werden mit der üblichen Funktionalität wie bei Texteditiersystemen unterstützt. Die Beiträge können abgelegt, zwischengespeichert oder auch gelöscht werden.

Der ***Zeitrahmen*** wird durch den Aufgabenbereich der Redaktionsgruppe diktiert. Dokumente, die keine Endfassung darstellen, werden nach der Redaktionsperiode gelöscht oder ggf. archiviert.

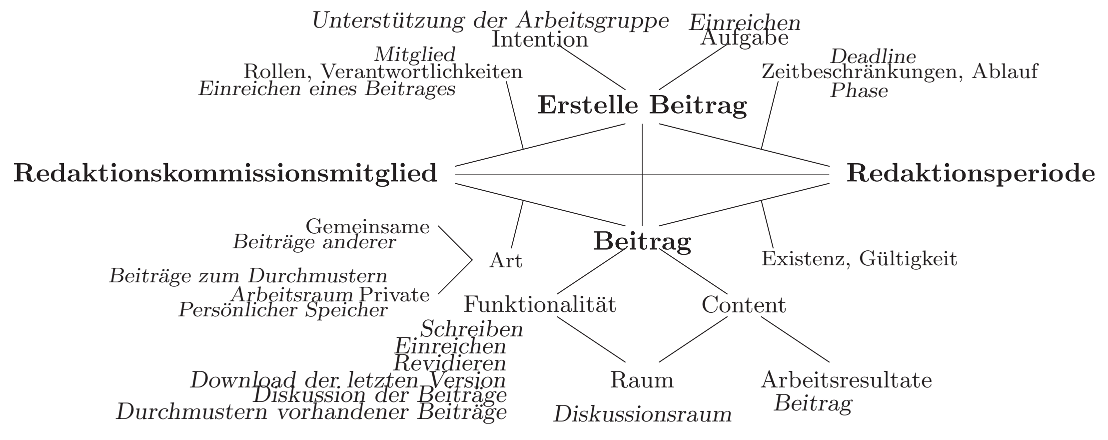


Bild 50: Der Spezifikationsrahmen für Beiträge von Arbeitsgruppenmitgliedern

Der Spezifikationsrahmen ist eine Verallgemeinerung der Theorie der Wortfelder.

Die detaillierte Spezifikation der Dialogschritte und Dialogszenen

Eine vollständige Beschreibung der Dialogschritte kann mit dem Arbeitsblatt erstellt werden.

Dialogschritt	
header	
Name	Layout
Titel	
Container	
Inhalt	
Text	
multimedia object	Graphik
	Bild
	Video
	Audio
Funktionalität	Operationen
	algorithmisches Objekt
Anpassungsstil	
Einordnung in Hierarchien	
Adhäsion	
Adaptation	
Interaktionsstil	
Steuerungstil	
involvierte Akteure	

Oft wird eine vollständige Beschreibung schwierig. Deshalb können wir eine Beschreibung gliedern in

obligatorische Bestandteile, deren Spezifikation unbedingt erforderlich ist,

weitere sinnvoll Bestandteile, (good practice) deren Spezifikation der weiteren Bearbeitung zugute kommt und die in einer Spezifikation nicht fehlen sollten,

optionale Bestandteile, die eine Beschreibung sinnvoll ergänzen, aber die nicht für den Abschluß der Spezifikation erforderlich sind, und

nützliche Bestandteile, die eine Einordnung und eine Beschreibung des Kontextes erlauben.

Diese Untergliederung erscheint auf dem ersten Blick als überfrachtet. Da in der Praxis Entwicklungsgruppen sehr häufig innerhalb kurzer Zeiträume wechseln bzw. je nach Projektetappe nur für eine kurze Zeit existieren, ist eine gute, alle Aspekte umfassende Dokumentation erforderlich.

Eine Beschreibung der Dialogszenen, in denen diese Untergliederung vorgenommen ist, wird im folgenden Arbeitsblatt angegeben:

Szene			
header			
Inhalt Problemgebiet Lösung	Name Intention	Entwickler Motivation auch bekannt als	copyright Source siehe auch
Variante	Anwendungsgebiet		
Anwendung			
Anwendbarkeit	Konsequenzen der Anwendung	Beispieleinsatz	angewandt in Anwendungen
Benutzbarkeitsprofil	Erfahrunsberichte	DBMS	
Beschreibung			
Strukturierung: Struktur, statische IC	Funktionalität: Operationen, dynamische IC, Erzwingungsstrategien	Interaktivität: Story-Raum, Akteure, Content-Objekte, Repräsentation	Kontext: Aufgaben, Intention, Geschichte, Umgebung, Ziele
Implementation			
Implementation	Programmkode	assoziierte Szenario	
assoziierte Szenen	Kollaboration	Integrationsstrategie	
`obligatorisch`	`good practice`	`optional`	`nützlich`

Ein Szenario ist z.B. in Bild 51 beschrieben.

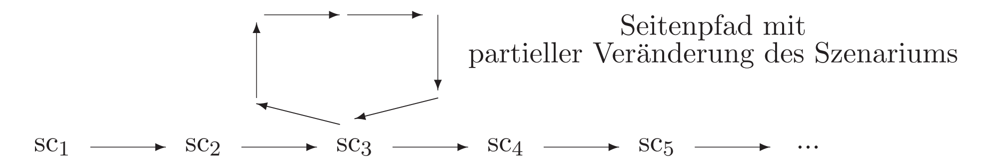


Bild 51: Szenario in einem Story-Raum

Ein Szenario ist durch eine Zuweisung von Werten an die Parameter konkretisiert. Damit wird das Szenario für einen Benutzer zu einem Szenarium, das dieser durchläuft. Mit der Zuordnung eines konkretisierten Szenario zu einem Benutzer wird damit auch der Akteur personalisiert.

Die Adaption der Elemente des Story-Raumes an einen konkreten Durchlauf kann durch den Aufbau von Sitzungsobjekten in der folgenden Form erfolgen. Sitzungsobjekte selbst verfügen wiederum über eine Historie und erlauben damit eine Aufzeichnung der Historie der Benutzung durch einen Akteur.

Ein Beispiel einer Lernszene wird in Bild 52 dargestellt. Ein Lehrstuhl erarbeitet das Lehrveranstaltungsangebot gemeinsam. Dabei existieren zwei Einwahlstränge, die parallel ablaufen: die Planung von Vorlesungen mit den Übungen etc. und die Erarbeitung eines Seminarvorschlages. Bei der Planung von Vorlesungen kann man auswählen, ob eine Anforderung bearbeitet wird oder ob ein neuer Kurs erarbeitet wird.

Bei der Eingabe von Daten kann man auch auf alte, historische Daten zurückgreifen. Analog kann auch ein Mitarbeiter eines Lehrstuhles seine Arbeitsaufgaben diskutieren. Am Ende werden die Daten durch den Lehrstuhlleiter eingereicht.

Eine analoge Szene können wir auch generisch entwickeln. Eine Suchszene in einer Webseite muß unterschiedliche Facetten der Suche darstellen:

- Die eigenschaftsbasierte Suche orientiert sich auf Haupteigenschaften, die auch als solche für den Content spezifiziert sein müssen z.B. durch Angabe von Schalen eines Sterntyps. Die eigenschaftsbasierte Suche muß robust sein. Wir wenden deshalb dafür SoundEx-Algorithmen an.

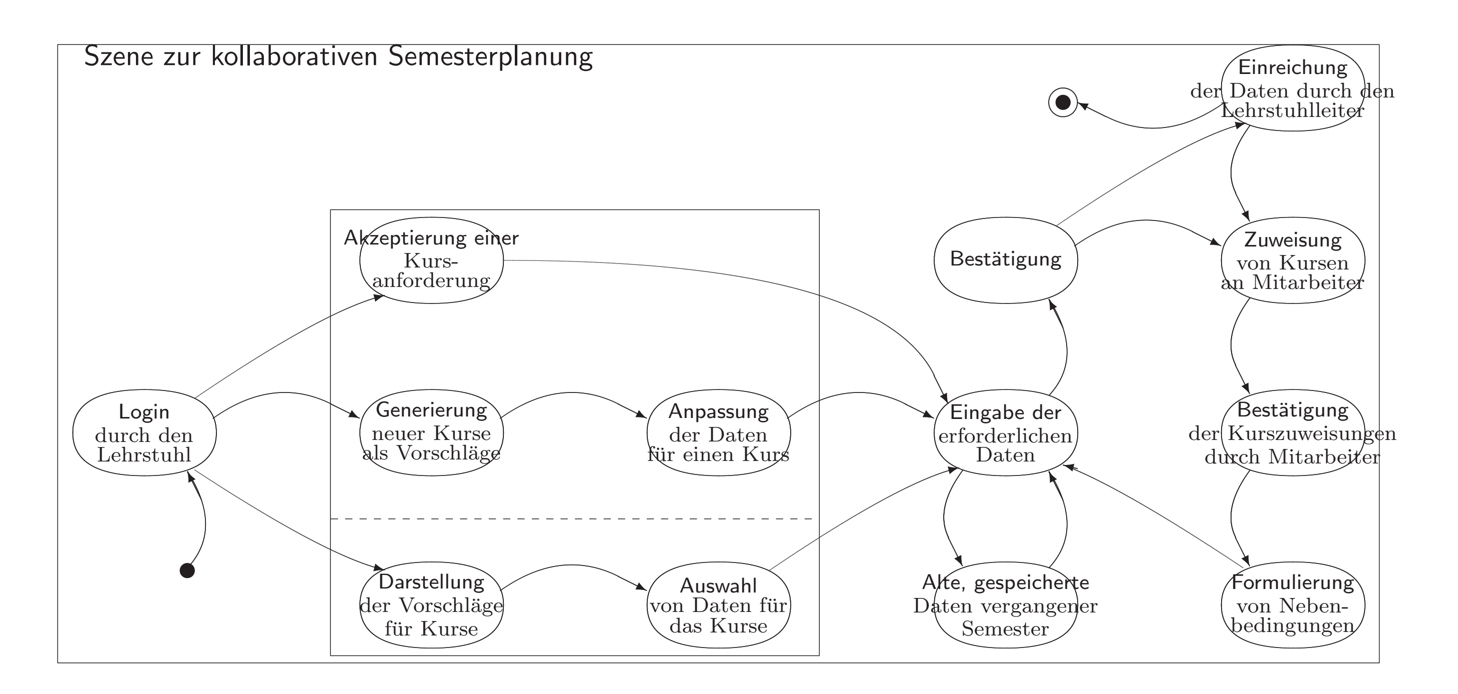


Bild 52: Szene zur kollaborativen Planung eines Lehrveranstaltungsangebotes eines Lehrstuhles

- Die assoziative Suche geht dagegen von assoziierten Begriffen aus. So kann z.B. eine Hotelsuche mit einer Suche nach Hotels in der Nähe von einer Sehenswürdigkeit oder eines Transportmittels beginnen, wobei die Nähe selbst durch Fuzzy-Funktionen unterstützt wird.
- Eine Suche kann auch für Schnäppchensucher von Sonderangeboten angeboten werden.

Die Suche ist ein hochgradig iterativer Prozeß mit einer schrittweisen Verfeinerung. Abschließend kann eine Buchung erfolgen. Die Suchszene kann zu jeder Zeit ohne weitere Schritte verlassen werden. In Bild 53 wird eine Suchszene dargestellt, die diese Aspekte umfaßt. Diese Gestaltung der Suchszene hat sich bei der Gestaltung von Websites bewährt. Mit dieser Gestaltung verwenden wir Techniken der *aspekt-orientierten* und *generativen Programmierung*. Gleichzeitig kann dieser Zugang als eine Variante der *subjekt-orientierten Programmierung* verstehen.

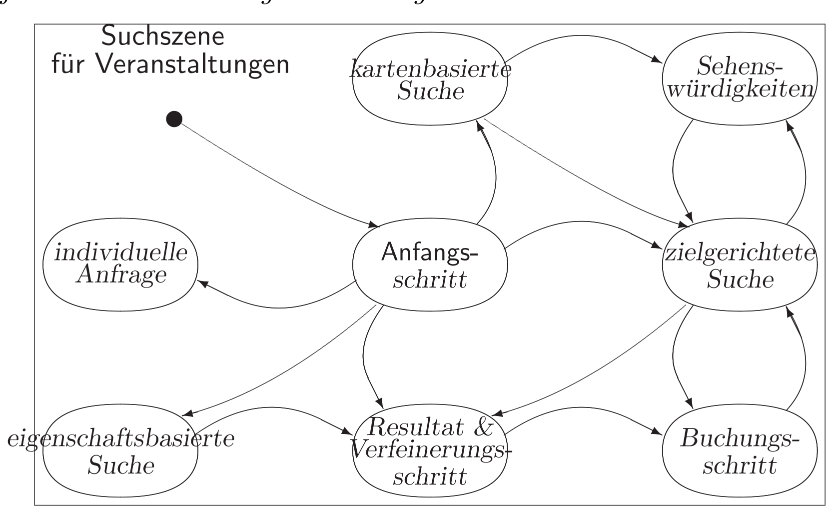


Bild 53: Dialogschritte innerhalb eines Suchszene

Neben den hier dargestellten Suchschritten gibt es noch weitere Varianten für Dialogschritte zur Suche:

- Die antonymbasierte Suche beginnt mit einem Begriff, den man nicht sucht, der aber relativ gut das Gegenteil umschreibt.

- Das “Zappen” erlaubt den Beginn an einer beliebigen Stelle und eine spezifische Form des “Drill-down”.
- Das “Roll-up” beginnt mit einem speziellen Begriff und hangelt sich über die Gattung oder Kategorisierung zu den gewünschten Begriffen.
- Das Umschauen oder Kramen ist eine Suche mit einer Drill-down-Funktion zur Verfeinerung.
- Die Formulierung eines vollständigen Suchausdruckes z.B. mit einer SQL-Anweisung ist eher die Ausnahme.
- Die Suche mit intelligenten, sich “durchfragenden” Agenten ist eine Form des Auslegens von Fallen oder der Beauftragung von Suchhelfern.

In analoger Form kann auch die Navigation oder auch der Export/Import in generischer Form dargestellt und mit konkreten Datenstrukturen unterlegt werden.

Der Kontext-Raum

Die Laufzeit-Präsentation wird durch Instantiierung des Kontextes (technische Umgebung, Aufgabe, Geschichte, Umstände) und durch Adaption an den Benutzer (Profil, Portfolio) erzeugt. Diese Information muß deshalb im Entwurf mit erarbeitet werden.

Wir betrachten unterschiedliche Spielarten von Kontext. Diese Spielarten können mit dem Zwiebelprinzip zum Ausspielen in die XML-Dokumente eingebracht werden.

Allgemeiner Kontext dient zur Beschreibung des Kontextraumes.

- Umstände allgemeiner Art kennzeichnen insbesondere Beschränkungen der Benutzung, Einspielen von Hilfsmitteln etc.
- Das Benutzungsmodell der Akteure hängt von einer Reihe von Parametern ab wie
  - die Bezahlung,
  - das Organisationsmodell zur Benutzung,
  - die daraus resultierenden Rechte und
  - die darauf aufbauenden Rollen.
- Das Portfolio der Akteure wird bestimmt
  - durch die Aufgaben,
  - durch die spezifischen Rechte,
  - durch die spezifischen Rollen,
  - durch die Umstände der Benutzung und
  - durch die Ziele.
- Technische Einschränkungen allgemeiner Art erweitern oder schränken die Benutzung ein. Sie sind gegeben durch
  - die Umgebung der Benutzung wie z.B. Hardware, Server-Software, Client-Software und den Kanal, sowie durch
  - die Verteilung auf unterschiedliche Knoten.

Der konkrete Benutzungskontext basiert auf einer Beschreibung der Assoziationen, wobei auch eine entsprechende Bindung, Umordnung zur sequentiellen Repräsentation berücksichtigt wird, und der Ort und die Zeit der Benutzung zu Veränderungen führen kann. Der Benutzungskontext ist determiniert durch

- die Einbettung in den Story-Raum insbesondere unter Berücksichtigung
  - des benutzten Content je nach angeforderter

Navigation u.a. Funktionalität, sowie dem
Sicherheitsprofil,
die Anpassung an den Benutzer, wobei auch
Ort,
Zeit und
Benutzungsgeschichte variieren können,
die Auslieferung von Content in Containern, deren Typ variieren kann und die auch an die
verpackten Content-Objekte
anpaßbar sein müssen,
durch das aktuelle Szenarium und die unterstützenden Session-Objekte,
das konkrete Benutzungsmodell,
die aktuelle Umgebung wie z.B.
den Kanal mit seiner aktuellen Übertragungskapazität und seiner Sicherheit, sowie die
aktuell gewählte Verschlüsselung.

Die Spielarten von Kontext können einer Abhängigkeitsstruktur unterliegen. Wenn wir z.B. voraussetzen,

- daß der syntaktische Kontext, der durch den Content bestimmt ist, und der Zusatzkontext, der durch die Hilfsmittel bestimmt ist, unabhängig voneinander sind und
- daß sich die Spielarten schichten lassen aufgrund der Abhängigkeitsbeziehung,

dann kann ein Ausspiel des Content in der Form erfolgen wie in Bild 54.

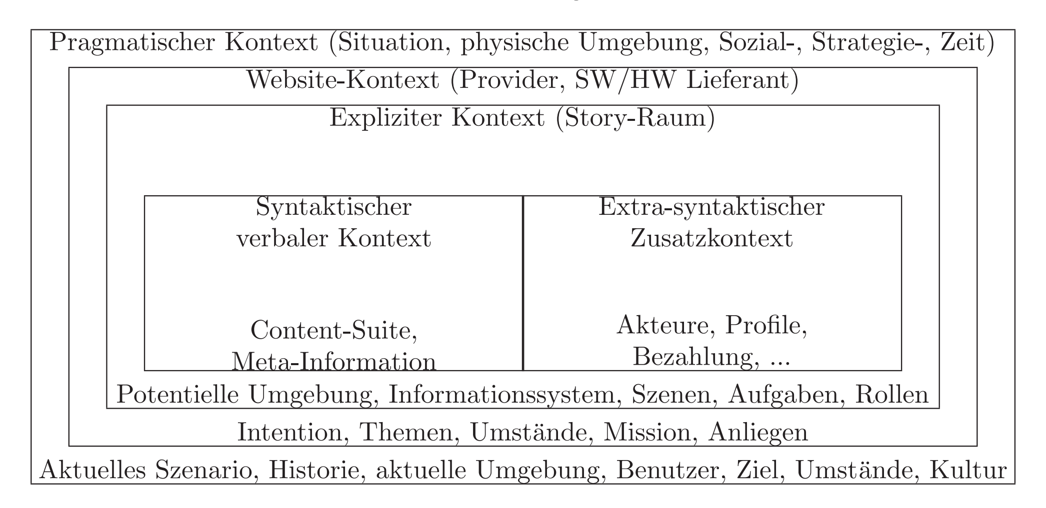


Bild 54: Das Zwiebelprinzip zum Einbringen von Kontext

Der Kollaborationsrahmen

Wir unterscheiden zwischen Kooperation, Koordination und Kommunikation. Diese drei Formen der Kollaboration werden im nächsten Abschnitt im Detail behandelt. Kollaboration zwischen Akteuren wird im Rahmen der Spezifikation des Story-Raumes und im Rahmen der Spezifikation der Verteilung dargestellt. Es werden unterschiedliche Aspekte für den Story-Raum dargestellt. Hier sind die allgemeinen Aspekte von Bedeutung. Für die Spezifikation der Verteilung werden diese Aspekte verfeinert und im Detail angegeben.

Der Kooperationsrahmen soll bei der Inszenierung eine automatische Generierung der Oberflächen für die einzelnen Dialogschritte unterstützen. Im einzelnen ist der Kooperationsrahmen spezifiziert durch Angaben zu:

Koordination bzw. Kooperation: Wir unterscheiden zwischen Koordination und Kooperation.

Charakterisierung nach Koordinationsformen: Eine Koordination von Akteuren erfolgt für die Bewältigung von Arbeitsaufgaben. Diese Aufgaben werden mit einer Reihen von Koordinationstypen verbunden. Typische Koordinationstypen sind z.B. die Broker- bzw. Trader-Customer-Koordination, die Client-Dispatcher-Koordination oder die Publisher-Subscriber-Koordination. Sie stellen allgemeine *entfaltbare Workflows* dar, bei denen der Ablauf der Koordination durch entsprechende verfeinerbare Dialogschritte gekennzeichnet wird. Diese Koordinationstypen werden im weiteren zum Austauschrahmen zur Spezifikation der Verteilung erweitert. Der Austauschrahmen umfaßt die gesamte Kollaboration.

Charakterisierung nach Kooperationsformen: Kooperation zwischen Akteuren basiert auf einer Darstellung des Arbeitsprozesses, einer Angabe des Organisationsmodelles und einer Darstellung des Arbeitsplatzes bzw. Arbeitsraumes.

Die Kooperationsformen zum Erreichen der Ziele werden im Rahmen der Kooperation der Benutzer abgeglichen. Es sind sowohl spezifische Formen der Interaktion als auch des Reviewing und der Kontrolle zu vereinbaren.

Die Rollen bei der Kooperation werden für die einzelnen Benutzer im Detail festgelegt.

Charakterisierung der Formierung: Wir unterscheiden unterschiedliche Arten der Formierung von Gruppen in

inhaltsbezogene Formierung, bei der der Kooperationsrahmen durch Ziel und Portfolio determiniert wird,

arbeitsweise-orientierte Formierung, die eine Anpassung der Content-Objekte an die z.Z. präferierte bzw. im nächsten Schritt erwartete Arbeitsweise ermöglicht, und

Formierung durch Selbstorganisation der Gruppe, die eigenständig die Inhalte, den Zeitpunkt und den Arbeitsraum bestimmt.

Charakterisierung nach Raum und Zeit: Wie bereits dargestellt, können wir bei einer Zusammenarbeit unterschiedliche Content-Objekt-Zuordnungen und Zeiträume darstellen.

Gleiches Content-Objekt und synchrone Zusammenarbeit: Ein typische Form dieser Zusammenarbeit sind *Brainstorming-Sitzungen*.

Gleiches Content-Objekt und asynchrone Zusammenarbeit: Ein typische Form dieser Zusammenarbeit kann man in *Videokonferenzen* beobachten.

Verschiedene Content-Objekte und synchrone Zusammenarbeit: *CASE-Werkzeuge* realisieren diese Art der Zusammenarbeit.

Verschiedene Content-Objekte und asynchrone Zusammenarbeit: Diese Zusammenarbeit ist z.B. für *elektronische Post* typisch.

Charakterisierung nach Kooperationsvertrag: Der Kooperationsvertrag dient dem Abgleich der Interessen der kooperierenden Benutzer. Es werden sowohl Absprachen zu den Inhalten als auch zu den zu wählenden Szenario sowie die Einordnung in Arbeitsräume und Zeiträume getroffen.

Art des Zusammenwirkens: Kollaboration basiert auf einer expliziten oder impliziten Kommunikation, auf Regeln des Zusammenwirkens und einer Dramaturgie des Zusammenwirkens. Die Art des Zusammenwirkens wird oft in kanonischer Form vorgegeben. In diesem Fall wird das Zusammenwirken durch kleine Szenario bestimmt, die miteinander kombiniert werden können.

Die Art des Zusammenwirkens wird oft mit einem *Vertrag* der Kollaboration gekoppelt. Bestandteile des Vertrages sind die klassischen juristischen Fallfragen:

“Wer”

arbeitet zusammen (***“wie”***)

“mit wem”

zu welchem Gegenstand (***“was”***)

auf welcher Anspruchsgrundlage (***“woraus”***)

Diese Fallfragen verallgemeinern wir zu Spezifikationsrahmen für die Art des Zusammenwirkens:

Die Beziehungen der Anwender und die Beziehungen des Benutzers mit dem System können durch *Beziehungsstrukturen* dargestellt werden. Diese Beziehungsstrukturen können Ethikregeln unterliegen oder explizit formuliert sein. Wir können wie im Falle der aspektorientierten Programmierung auch auf allgemeine Beziehungsstrukturen zurückgreifen oder explizit die Einhaltung der *allgemeinen Geschäftsbedingungen* postulieren.

Arten der Aktivität werden durch Verbgruppen mit Verben der Handlungen wie *kaufen*, *lernen* und *informieren*, ergative Verben wie *fliehen*, Prozeßverben wie *einschlafen* (ingressive Prozesse) und *verblühen* (egressive Prozesse) und Verben zur Beschreibung eines Zustandes wie *schlafen* oder *haben* relativ gut charakterisiert. Wir sind für Informationssysteme an der ersten und der letzten Verbgruppen stärker interessiert. In diesen beiden Gruppen unterscheiden wir

- Verben des Geschehens,
- Verben des Zunehmens,
- Verben der Übereinstimmung/Verschiedenheit,
- Verben des Mitteilung,
- Verben des Argumentation,
- Verben der Zustimmung,
- Verben der Leitung,
- Verben des Zusammentreffens,
- Verben der sinnlichen Wahrnehmung,
- Verben der Nahrungsaufnahme und
- Verben des Reinigens.

Die ersten acht Gruppen sind für Informationssysteme relevant und können zu speziellen Kooperationsrahmen verwendet werden.

Diskurstypen in der Zusammenarbeit können nach der Konversationstheorie unterschieden werden in:

Handlungen: Es wird der Partner zu einer Handlung aufgefordert.
Klärung: Es erfolgt mit dem Partner eine Klärung.
Entscheidung: Es wird mit dem Partner eine Entscheidung getroffen.
Orientierung: Es wird dem Partner eine Orientierung für dessen Tätigkeiten gegeben.

Wir können mit dieser Klassifikation der Arten des Zusammenwirkens in Erweiterung der Klassifikation *Wer-2-Wem* (2 steht für “to” (mit)) mit einem Muster der Form

$$\text{Provider}^{\textit{Art des Inhaltes}}\ 2\ \text{Kunde}^{\textit{Art der Aktivität}}$$

charakterisieren. Daraus ergibt sich für die Gestaltung von Websites eine Klassifikation in:

E-Business-Sites: $\mathbf{B(usiness)}^{\mathcal{P}(rodukt)}\mathbf{2V(erwaltung)}^{\text{kaufen}}$, $\mathbf{B}^{\mathcal{P}}\mathbf{2B}^{\text{kaufen}}$, $\mathbf{B}^{\mathcal{P}}\mathbf{2K(unde)}^{\text{kaufen}}$, $\mathbf{V}^{\mathcal{I}(nformation)}\mathbf{2K}^{\text{kaufen}}$, $\mathbf{K}^{\mathcal{P}}\mathbf{2\ K}^{\text{kaufen}}$ (bzw. $\mathbf{C(ustomer)}^{\mathcal{P}}\mathbf{2\ C}^{\text{kaufen}}$)

Lern-Sites: $\mathbf{B}^{\mathcal{W}(issen)}\mathbf{2K}^{\text{lernen}}$, $\mathbf{K}^{\mathcal{W}}\ \mathbf{2\ K}^{\text{lernen}}$

Information-Sites: $\mathbf{B}^{\mathcal{I}(nformation)}\mathbf{2G(ast)}^{\text{informieren}}$, $\mathbf{V}^{\mathcal{I}}\mathbf{2G}^{\text{informieren}}$, $\mathbf{K}^{\mathcal{I}}\mathbf{2G}^{\text{informieren}}$

Arbeitsgruppen-Sites: $\mathbf{A(rbeitsgruppe)2A}^{\text{agieren}}$, $\mathbf{A2G}^{\text{informieren, agieren}}$

Corporate-Identity-Sites: $\mathbf{P(rovider)}^{\mathcal{I}}\mathbf{2G}^{\text{anschauen}}$

Unterhaltungs-Sites: $\mathbf{B}^{\mathcal{U}(nterhaltung)}\mathbf{2G}^{\text{agieren}}$, $\mathbf{G}^{\mathcal{U}}\mathbf{2G}^{\text{agieren}}$

Arbeitsplatz: Der Content-Typ Arbeitsplatz soll die Kollaboration unterstützen. Er muß deshalb auch die Aspekte der Kollaboration berücksichtigen. Zur Darstellung benutzen und verfeinern wir die Kern-Typen des Content-Typs Arbeitsplatz:

Akteure werden mit ihren Kollaborationsbeziehungen dargestellt. Sie umfassen

Kooperationsbeziehungen,
Koordinationsbeziehungen und

Kommunikationsbeziehungen, sowie das
Organisationsmodell.

Gruppen verfügen über spezifische Formen der Kollaboration. Diese Kollaboration basiert oft auf relativ festgeschriebenen und demzufolge abzubildenden Beziehungsstrukturen.

Rechte werden mit der expliziten Darstellung der Kollaboration in Rechte zu Kooperation, Koordination und zur Kommunikation untersetzt.

Portfolio werden den Einzelaufgaben zugeordnet, wobei die Art des Zusammenwirkens auch die Art der Abarbeitung des Portfolios determiniert.

Die Organisation wird durch die Darstellung der Dramaturgie der Kollaboration verfeinert.

Der Kollaborationsrahmen wird noch einmal bei der Spezifikation der Verteilung betrachtet, dort allerdings mit einer Konzentration auf die technische Unterstützung. Zu Spezifikation der Kollaboration können wir die folgende Tabelle oder auch ein Arbeitsblatt wie bereits bei der Spezifikation der Szenen und der Dialogschritte verwenden:

Kollaborationsrahmen												
Kollaboration					Art des Zusammenwirkens			Arbeitsplatz				
Form	Rollen	Formierung	Raum / Zeit	Vertrag	Beziehungen	Arten	Diskurstyp	Akteure	Gruppe	Rechte	Portfolio	Organisation
...	...	...	...	...	...	...	...	...	...	...	...	...

Wir unterscheiden die *Kopplungsmechanismen* nach Seite 98 in *Interaktionskopplung* (Kopplung im Story-Raum) *Komponentenkopplung* (Container-Kopplung) und *Vererbungskopplung*. Sie können auch im Kombination verwendet werden.

Die *Kohäsion* kann analog zur Kohäsion auf Seite 98 durch die Bindung zwischen den einzelnen kooperierenden Objekten beschrieben werden.

Der Gestaltungsrahmen

Durch die Spezifikation eines Gestaltungsrahmens kann in allgemeiner Form die Darstellung der gesamten Arbeitsoberflächen-Suite und des Präsentationsraumes in einheitlicher Form und auch mit der XML-Technologie erfolgen.

Zur Gestaltung von Software und insbesondere von Dialogen nach ergonomischen Kriterien stellt die DIN-Norm 66234, Teil 8 folgende Kriterien auf:

Erwartungskonformität: Ein Dialog ist erwartungskonform, wenn sich die Erwartungen, Erfahrungen und bisherigen Handlungen der Benutzer im Dialog widerspiegeln.

Steuerbarkeit: Ein Dialog ist steuerbar, wenn er sich dem augenblicklichen Arbeitsstil, der Geschwindigkeit und der Wahl der Arbeitsmittel anpassen läßt.

Aufgabenangemessenheit: Ein Dialog ist aufgabenangemessen, wenn er die Erledigung der Arbeitsaufgaben unterstützt, ohne zusätzlich zu belasten.

Selbstbeschreibungsfähigkeit: Ein Dialog ist selbstbeschreibungsfähig, wenn er entweder direkt oder indirekt (z.B. über adäquate Hilfen) verständlich ist.

Fehlerrobustheit: Ein Dialog ist fehlerrobust, wenn trotz erkennbarer Fehler ein richtiges Resultat erzeugt wird.

Bei der Bewertung von Benutzungsoberflächen können eine Reihe von Parametern betrachtet werden:

Robustheit: Ein System darf durch eine falsche Handlung nicht in seinen wesentlichen Parametern (Benutzbarkeit, Durchlauf etc.) beeinflußt werden.

Benutzbarkeit: Die Benutzbarkeit bzw. Brauchbarkeit kann durch verschiedene Parameter bewertet

Analytische Meßmethoden werden z.B. beim Vollständigkeitstest herangezogen. Damit kann ermittelt werden, ob alle benötigten Informationen auch dargestellt werden.

Leistungsparameter sind z.B. die benötigte Arbeitszeit, die Fehlerrobustheit und die Zeitersparnis.

Die *kognitive Beanspruchung* stellt die geistige Anstrengung des Benutzers dar. Stimmen das mentale Modell des Benutzers und die Reaktionen am Bildschirm überein, dann ist sie gewöhnlich gering.

Die *Benutzerzufriedenheit* berücksichtigt die Nützlichkeit des Systemes für den Anwendungsbereich und auch den Lernaufwand zur Bedienung des Systemes.

Die Modellierung von Benutzungsoberflächen umfaßt die Spezifikation verschiedener Bereiche:

Dialogmethoden erfüllen unterschiedliche Zwecke. Wir können verschiedene Zugänge in realen Systemen finden:

Eingabemasken entsprechen Formularen. Damit ist auch die Arbeitsweise determiniert.

Befehle und Aufforderungen an den Benutzer werden zum Abruf von Daten benutzt. Dabei kann die Struktur durch meist deterministische Ablaufdiagramme dargestellt werden.

Menüs können mehrere Optionen oder Funktionen, aus denen der Benutzer auswählen kann, darstellen. Menüs können auch als Popup- oder Klappmenüs gestaltet werden.

Schaltflächen dienen zum Auslösen von Funktionen.

Die *Eingabemaske* eignet sich am besten zur Eingabe von Informationen über eine semantische Einheit. Sie sind weniger für die Modifikation von Daten geeignet. Dazu ist eine Menüstruktur eher geeignet.

Arbeitssicht: Eine Arbeitssicht (auch Arbeitsbereich in der Literatur) definiert alle Informationen, die für eine bestimmte Aufgabe benötigt werden.

Layout: Durch das Layout wird die Darstellung der Informationen der einzelnen Arbeitsschritte beschrieben.

Prozeßsicht: Durch die Prozeßsicht wird der Arbeitsablauf bzw. der Geschäftsprozeß der Anwendung spezifiziert.

Diese ergonomischen Prinzipien, die allgemein für Softwareprodukte entwickelt wurden, können zu Gestaltungsprinzipien weiterentwickelt werden bzw. von produktspezifischen Forderungen kann abgesehen werden. Wir führen als allgemeines Rahmenwerk Gestaltungsrahmen ein.

Die Gestaltung von Schnittstellen besitzt eine Reihe von Analoga mit der Gestaltung von Werbematerial. Aus Optimierungsgründen sind jedoch kaum dessen gestalterische Möglichkeiten ausschöpfbar. Insbesondere sind die Effizienz und die Übersichtlichkeit zu beachten. Hinzu kommen aber auch zusätzliche Möglichkeiten. Werkzeuge, die z.Z. im Entstehen sind, werden auch Sprache, Gestik, intelligente Agenten und integrierte Multimedia (Panmedia ist ein besserer Begriff) einschließen.

Graphiken von Web-Oberflächen werden immer noch mit Blick auf unbegrenzte Ressourcen entworfen. Mit dem Internet II aber auch schon bei einer Vermittlung von Informationen über Modems sollte man sich bei der graphischen Gestaltung auf die Informationsvermittlung besinnen und auf graphische Arabesken und Manierismen sowie intergalaktische Multimedia-Effekte verzichten.

Oberflächen sollten im allgemeinen der *Anwendungsphilosophie*, der *Anwendungslogik* und dem *Anwendungszweck* folgen. Deshalb sollte eine Anwendung immer als Ganzes entworfen werden. Die *Organisation* der Oberflächen und die *visuelle Struktur* der einzelnen Oberfläche folgen der Anwendungslogik. Sind die Oberflächen zur Präsentation bestimmt, dann ist auch die Firmenstrategie mit einzubeziehen. Das Corporate Design - von der Werbung bei der Beratung teuer als das Entwurfswissen eingebracht - ist nicht von der Darstellung zu trennen. Bestimmte Bedienelemente wie z.B. die rechte Maustaste können für spezielle Effekte in einer ganzheitlichen Gestaltung reserviert werden.

Die Informationsdarstellung, die Darstellung des Arbeitsprozesses und die davon abhängige Darstellung der Suchmechanismen sollte zum einem integriert erfolgen, zum anderen durch die Architektur

in separaten Einheiten gehalten werden. Insbesondere sollten der Suchmechanismus und die verschiedenen Verknüpfungsnetze nicht mit der Information gemeinsam dargestellt sein. Eine Integration bedeutet keinesfalls die weitestgehende *Unabhängigkeit* der einzelnen Oberflächen voneinander aufzugeben. Verbindungen zwischen den einzelnen Oberflächen sind explizit darzustellen bzw. durch globale Techniken wie *Zwischenablagen* und dynamischen Datenaustausch zu unterstützen. Kontextsensitive Oberflächen, insbesondere solche, die von mehreren anderen abhängen, sollten vermieden werden oder nur mit einer hierarchischen Strukturierung angewandt werden.

Eine Darstellung von Informationen sollte so einfach[14] sein, wie es die Informationsfülle zuläßt. Die Information ist *übersichtlich* zu gestalten. Durch geschickte Vernetzung von Oberflächen kann eine Übersichtlichkeit geschaffen werden, die weit über die Möglichkeiten von Printmedien hinausgeht. Durch verschiedene übergreifende Verzeichnisse kann eine Transparenz geschaffen werden, die eine umfassende und aktuelle Recherche in einfacher Form ermöglicht. Einfachheit impliziert auch *Eleganz*. Der *Stil* ordnet sich hier unter. Die *Repräsentation* und das *Aussehen* folgen auf dieser Grundlage.

Einfache Oberflächen bedeuten auch *minimale Wege* sowohl für die Bedienung als auch für das Auge. Mengen von Oberflächen werden umso eher angenommen, umso mehr sie einer *einheitlichen Strukturierung* unterliegen. Die Verteilung der Funktionalität sollte einheitlich sein. Die *Eingabeoberflächen* benötigen eine einfache und übersichtliche Gestaltung. Sind aufgrund einer Informationsvielfalt mehrere Oberflächen notwendig, dann ist auch der Zusammenhang explizit darzustellen.

Die Informationsdarstellung muß klar, einfach und intuitiv verständlich sein. Negative Information und negative Anfragen erfordern vom Benutzer ein genaues Verständnis der unterlegten Logik. Besser ist es, diese Information in *positiver Logik* zu formulieren. *Farben* können Information tragen. Sie sollten aber stets der Informationsdarstellung untergeordnet werden. In verteilten Anwendungen sollte man mit Farben sparsam umgehen und den Schwarz-Weiß-Test nicht auslassen. *Warnhinweise* sollten auch als solche unmißverständlich zu erkennen sein. *Fehlermeldungen* sollten kontextsensitiv, minimal und auch für den Normalbenutzer intuitiv verständlich sein. Die Darstellung von wesentlichen Informationen sollte *plattformunabhängig* erfolgen. Die *Statusleiste* kann auch eine Kurzhilfeinformation mit einbeziehen. *Skalierung, Kontrast* und *Größenverhältnisse* sind der Informationsdarstellung zu unterwerfen.

Oberflächen sollten an die Fertigkeiten und Fähigkeiten der Benutzer angepaßt sein. Die Benutzer können nach den Kriterien von [Alt] charakterisiert werden: positive Erfahrungen (wie z.B. Arbeitssprache), negative Erfahrungen (z.B. Fehler, Entscheidungen, Wortwahl) und Fertigkeiten bzw. Fähigkeiten (z.B. Wissen, motorische, visuelle Fertigkeiten, Abstraktions- und Formulierungsfähigkeiten etc.). Damit hat auch die Orientierung auf den Benutzer Vorrang vor dem Testen von nichtstandardisierten Werkzeugen. Obwohl die Maus in vielen Anwendungen zum normalen Arbeitsinstrument geworden ist, sollte stets auch die *Tastatur* unterstützt werden. Sie ist meist schneller und bei sinnvoller Anordnung der einzelnen Arbeitsschritte kann sogar der Tabulator für das Beenden eines Teilschrittes und den Beginn des nächsten Teilschrittes benutzt werden. Sind unterschiedliche Gruppen von Benutzern zu berücksichtigen, dann sollte auch ein unterschiedliches *Bedienniveau* implementiert werden. Die *Bediensprache* ist dem Benutzer und seinem Anwendungsgebiet mit anzupassen. Jede Art von Benutzung führt zu einem *Feedback* durch das System. Damit wird einem Benutzer der nächste Schritt erleichtert.

Weiterhin determinieren die Fähigkeiten der Benutzer und ihren Bedürfnissen, ob ein *black-box-Zugang*, der dem Benutzer Details der Implementation vollständig vorenthält, ein *glass-box-Zugang*, der dem Benutzer auch gestattet, die Arbeitsweise des Systems, insbesondere das input-output-Verhalten, zu verstehen und sein Verhalten dementsprechend zu verändern, oder ein Mix dieser beiden Zugänge anzustreben ist.

Da eine Vielzahl von Oberflächen in einer Anwendung zu entwickeln sind, ist auch bei der Gestaltung die Wirtschaftlichkeit zu beachten. Die Oberflächen sollten *generisch* bzw. *parametrisch* sein oder zumindest einem *Standard* folgen, der mit der Anwendung korreliert. Die *Wiederverwendung* von existierenden Oberflächen erleichtert ebenso wie die Standardisierung das Erlernen der Benutzung.

Die Effizienz und Aspekte der Sicherheit sollten durch das Dialogmangementsystem (oder falls kein System existiert, dann sind diese Probleme beim Entwurf der Oberflächen mit zu betrachten)

---

[14] Vielleicht sollte man doch das "Small is beautiful" in den Mittelpunkt stellen.

optimierbar sein.

Die Gestaltung von Oberflächen erfordert die Einbeziehung so unterschiedlicher Disziplinen wie Wahrnehmungspsychologie, Ergonomie, Soziologie, Informatik, Grafikdesign und Marketing in die Datenbankprogrammierung. In der ersten Näherung können die Gestaltungselemente (Typographie, Symbole und Piktogramme, Farbe sowie Proportion und Aufbau des Bildschirms) betrachtet werden.Hinzu kommen die Betrachtung der Eleganz und der Einfachheit, der Organisation und visuellen

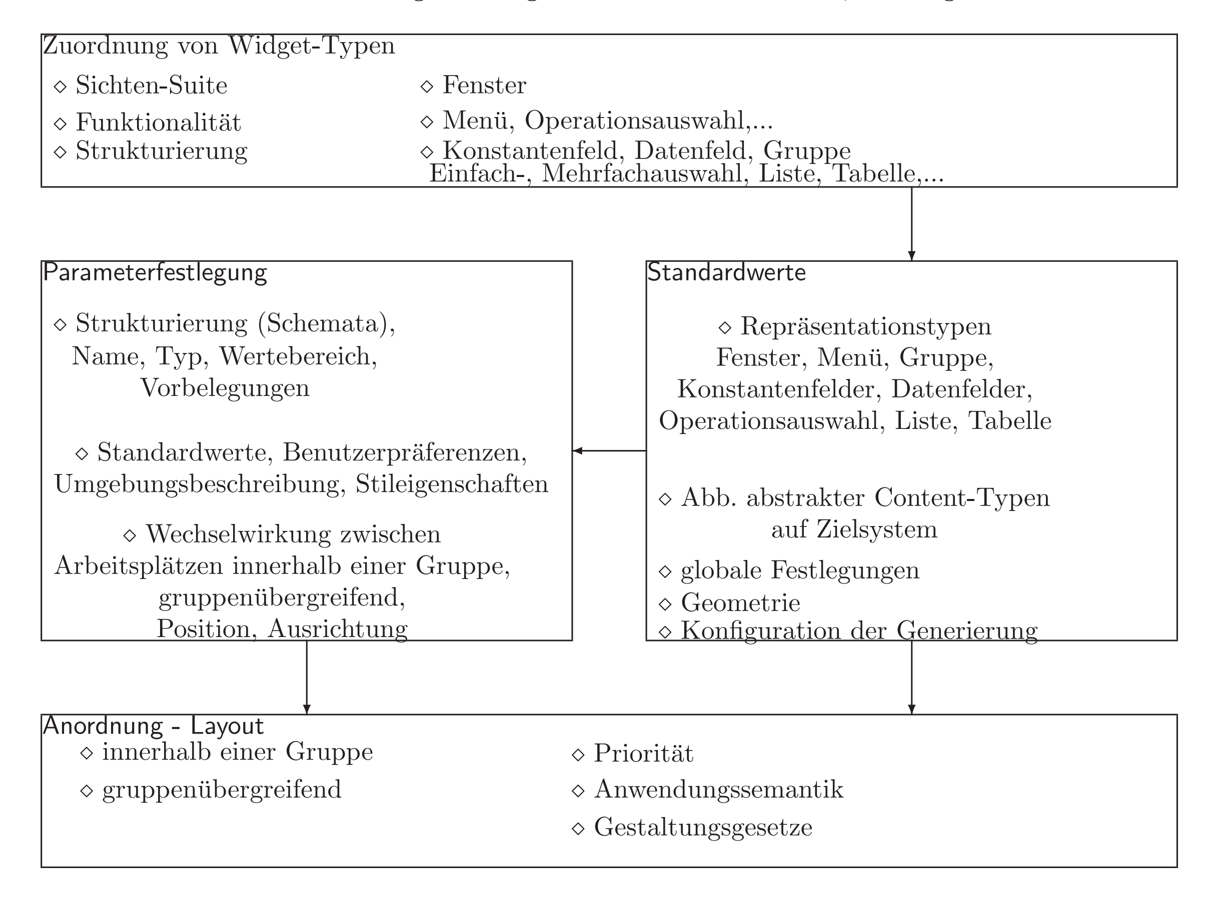


Bild 55: Die Vorgehensweise zur Zusammenstellung von Benutzungsoberflächen

Struktur, der Anordnung und Bedienung, der Bilder und der Repräsentation und Stilfragen. Auf dieser Architektur kann auch die Vorgehensweise zur Generierung von Benutzungsschnittstellen wie in Bild 55 aufgesetzt werden.

Der Gestaltungsrahmen soll uns eine allgemeine Beschreibung der Gestaltung erlauben und auch eine automatische Adaptierung oder Adaption an die Eigenarten der Benutzer, an die technische Umgebung und den Kontext im allgemeinen gestatten. Es sind bereits eine Vielzahl von Regeln zur Gestaltung von Graphischen Benutzungsoberflächen bekannt. Diese Regeln werden jedoch selten im Zusammenhang betrachtet. Mit der Spezifikation des Gestaltungsrahmens wollen wir jedoch auch den Zusammenhang in der Gestaltung betonen und zugleich auch eine Einheitlichkeit bei der Gestaltung realisieren.

Der Gestaltungsrahmen erlaubt auch eine allgemeine Kategorisierung der Gestaltung und zugleich auch eine Assoziation mit Metaphern, die als gesamtheitliche Metaphern der gesamten Gestaltung unterlegt werden können. Wir kommen auf diese Anwendung am Ende des Abschnitt noch einmal zurück.

Zuerst stellen wir unseren Zugang zum Gestaltungsrahmen vor. Der Gestaltungsrahmen ist durch

Playout mit Werkzeugen,

Layout

Metaphern,

Akteure und

Qualitätsanforderungen

spezifiziert.
Die unterschiedlichen Layoutentscheidungen werden in Monographien zu graphischen Benutzungsschnittstellen ausführlich behandelt. Für die Spezifikation des Gestaltungsrahmen wollen wir uns allerdings auf generelle Typen von Layoutentscheidungen konzentrieren. Deshalb wird nicht im Detail diskutiert werden, ob *grün* oder *pastellgrün* passende Farben sind. Die Charakterisierung der Akteure haben wir bereits bei der Spezifikation des Story-Raumes vorgenommen. Diese Spezifikation wird nun mit den Profilen und den Portfolios kombiniert. Die Qualitätsanforderungen haben wir bereits im Detail eingeführt. Wir werden sie jedoch im weiteren zu Qualitätsvorgaben für die Interfaces verfeinern.

Die einzelnen Komponenten des Gestaltungsrahmens sind die folgenden:

Werkzeuge zur Gestaltung der Benutzungsoberflächen in den 90er Jahren sind in einer Vielfalt entwickelt worden, daß eine Übersicht schwerfällt. Zugleich fällt ins Auge, daß fast alle diese Werkzeuge keine große Verbreitung gefunden haben. Eine Ursache ist sicherlich auch der Hang zur 'featuritis'.

Die Werkzeugentscheidung sollte sich an einer Klasse von Interfacewerkzeugen orientieren. Mit der Angabe von Gestaltungsrahmen werden sich auch stärker Typen von Interfacewerkzeugen herausschälen, wenn sich durch XML allgemeine Standards wie z.B. SVG zur Graphikdarstellung durchsetzen.

Der Typ von Interfacewerkzeugen wird durch eine Darstellung folgender Parameter spezifiziert:

Funktion: Die Werkzeuge können

- einfache Funktionen zur Verfügung stellen z.B. wie HTML mit einer Reihe von Umgebungen,
- komplexe Funktionen bereitstellen, mit denen z.B. auch ein Playout von Simulationen, analoge und digitale Video- und Audio-Dateien oder Kodierung, Fehlerbehandlung etc. unterstützt werden,
- Kommunikations-, Kooperations- und Koordinationsfunktionen sowie Austauschformate unterstützen,
- Bindungsfunktionen, mit denen Informationen und Content-Suiten eingebunden werden können, besitzen und
- Verwaltungsfunktionen zur Verwaltung und Weiterführung von Sitzungen und Arbeitsräumen anbieten.

Aufgabenklasse: Durch die Werkzeuge werden bestimmte Aufgaben unterstützt und ggf. auch nicht unterstützt. Es sind die Aufgabenklassen zu charakterisieren, die durch die Werkzeuge unterstützt werden.

Paradigma der Darstellung: Die einzelnen Funktionen der Werkzeuge erlauben unterschiedliche Darstellungen wie z.B. ereignisorientierte, objektorientierte oder zustandsorientierte Darstellung der Funktionsabarbeitung.

Kollaboration: Die Funktionalität der Werkzeuge kann ggf. auch einem Benutzer seinen Schreibtisch, einer Gruppe einen Arbeitsraum, eine Kollaborationsunterstützung oder auch Arbeitsspeicher bereitstellen. Die Kollaborationsunterstützung basiert auf einer Architektur zur Unterstützung der Kollaboration. Die Kollaboration erfordert ggf. auch langlebige Transaktionen und auch das Anlegen von temporären Klassen sowohl beim Benutzer als auch beim Server. Die Kollaboration kann über unterschiedliche Kanäle erfolgen.

Leistung: Werkzeuge stellen Funktionalität in einer bestimmten Qualität und mit einer bestimmten Kompetenz zur Verfügung. Die Spezifikation der minimalen Qualitätsanforderungen wie Allgegenwart und Sicherheit ist mit dem Gestaltungsrahmen vorzugeben.

Layout: Das Benutzungsinterface soll dem Akteur ein einfaches Agieren zur Bewältigung seiner Aufgaben gestatten. Es kann in allgemeiner Form die Art des Benutzerinterfaces durch einen Layout-Guide vorgegeben werden. Layout-Guides können sich an die 'corporate identity' des Betriebes anlehnen, können unterschiedlichen Gestaltungsrichtlinien folgen und auch durch entsprechende Regelwerke an den Kontext und die Benutzung adaptiert werden.

Die Gestaltung von Schnittstellen sollte den oben dargestellten Prinzipien der Ergonomie und der Psychologie folgen. Dazu gehören auch Prinzipien der visuellen Gestaltung.

Das Layout wird durch eine Spezifikation folgender Parameter vorgegeben:

Metapher: Ein System soll sich dem Benutzer in einer einheitlichen Form präsentieren, wobei die allgemeine Arbeitsumgebung ebenso wie eine bevorzugte Form der Darstellung mit einbezogen wird. In unserem Beispiel kann z.B. die *Raumplanung* mit einer Reiter-Darstellung, die *Vorlesungsübersicht* durch hierarchische Strukturen unterstützt werden, die dem *Studienplan* und der *Lehrstuhlübersicht* folgen, unterstützt werden.

Screen-Layout für Funktion und Interaktion: Funktionen und insbesondere Interaktionsfunktionen sind als besondere Gestaltungselemente durch eine entsprechende Typisierung einheitlich und schnell erkennbar gestaltet.

Umsetzung der Prinzipien der (visuellen) Wahrnehmung: Schnittstellen sollen einfach , leicht zu überschauen und auch so zu bedienen sein, daß die Übersicht nicht verloren geht. Dazu sind Parameter der visuellen Wahrnehmung wie Ordnung und insbesondere Hierarchie, Wirkung auf bestimmte Akteure und auch der Schrittfolge durch eine entsprechende Struktur zu unterstützen. Daraus kann sowohl die vertikale als auch funktionale Navigation abgeleitet werden.

Da auch multimediale Elemente eingebracht werden können spielt neben der visuellen Kommunikation auch die audio-basierte Kommunikation sowie auch andere Arten eine Rolle. Insbesondere für barrierefreie Systeme wird auf die anderen Kommunikationsmöglichkeiten zurückgegriffen.

Umsetzung der Prinzipien der (visuellen) Kollaboration: Die unterschiedlichen Facetten der Kollaboration werden durch einen generellen Rahmen der Szenen, durch eine Abfolge der Szenen und durch didaktische Regeln bei der Erschließung des Story-Raumes allgemein dargestellt. Didaktische Regeln fassen sowohl die Redundanz der Kollaboration als auch die Erwartungskonformität.

Ebenso wie für die Realisierung von Barrierefreiheit eine Unterstützung durch nicht-visuelle Kommunikation erforderlich.

Berücksichtigung der Gestaltungsgesetze des Bildschirmes: Ein Bildschirm ist eine zweidimensionale Oberfläche, mit dem evt. auch dreidimensionale Effekte erzielt werden können. Die Gestaltung der Bildschirmoberfläche muß Gestaltungsprinzipien wie

- dem *Prinzip der visuellen Kollaboration* in Abhängigkeit von den Arbeitsaufgaben, der Story und der Einfachkeit der Vermittlung,
- dem *Prinzip der visuellen Wahrnehmung* basierend auf einer Abstimmung von Anordnung, Wirkung und Gliederung und
- dem *Prinzip der visuellen Gestaltung* unter Berücksichtigung der Optik, der Ähnlichkeit, der Geschlossenheit, der Symmetrie, der Prägnanz und des Aufnahmeflusses

angemessen berücksichtigen.

Akteure sind Gruppen von Benutzern. Die generelle Spezifikation der Akteure wurde bereits in diesem Kapitel dargestellt. Für den Gestaltungsrahmen nehmen wir eine Kategorisierung der Akteure vor nach

Einordnung in Zielgruppen,

Polaritäten-Profil, insbesondere das psychographischen Profil und nach

Adaption an

Profil

Portfolio und
Benutzungsgeschichte etc. vor.
Es werden außerdem die Erwartungshaltung, die allgemeinen Gruppeneigenschaften, der Bildungs- und Arbeitshintergrund klassifiziert.

Qualitätsvorgaben: Informationssysteme sollen sich durch eine hohe Benutzbarkeit auszeichnen. Benutzbarkeit kann man auf Qualitätsanforderungen abbilden:

Verständlichkeit: Es sind alle Funktionen, die Navigation und Aufforderungen unmißverständlich für einen Benutzer.

Einfachheit Das System ist einfach gehalten. Das System lenkt nicht von der Lösung von Aufgaben ab.

Erlernbarkeit: Es soll einem Benutzer, der das System das erste Mal benutzt, und einem Benutzer, der das System nach längerer Pause wieder benutzt, ein einfacher Einstieg in die Benutzung des Systemes ohne hohen Lernaufwand möglich sein.

Diese Anforderungen kann man auf Merkmale des Systemes abbilden:

Erscheinungsform des Interfaces: Das Interface ist einfach, nicht überladen, besitzt ein ansprechendes Layout und eine akteurbezogene Inhaltsgestaltung.

Durchschaubarkeit des Story-Raumes: Der Story-Raum wird so optimiert, daß ein Benutzer seine Aufgaben mit dem einfachsten Szenarium bewältigen kann. Hilfreich ist hierbei eine angemessene Navigation im Story-Raum.

Les- und Browsbarkeit des Content: Der Content soll in einer Form präsentiert werden, die sowohl das Lesen, als auch das schnelle Durchmustern erlaubt, die keine Sprachbarrieren aufbaut (weder fremdsprachig noch von der Umgangssprache her) und die sprachlich einfach ist.

Vertrautheit und Nutzbarkeit: Ein Benutzer soll die Form der Benutzung und insbesondere die Szenario nicht erst erlernen, sondern sollte aufgrund seiner Erfahrungen das System einfach handhaben, einfach erlernen und schnell die Arbeit an beliebiger Unterbrechungsstelle wieder aufnehmen können.

Diese Merkmale können mit entsprechenden Metriken unterlegt werden.

Diese Forderungen wurden in der Cottbuser Arbeitsgruppe unter dem Begriff “omasicher” zusammengefaßt.

- Ein Informationssystem sollte ohne zusätzliche Ausbildung, in einfacher Art, aufgrund von offensichtlichen Optionen, für jedermann, innerhalb des Erwartungshorizontes des Benutzers, mit kontextsensitiver Hilfe, mit entsprechender Wortwahl, mit einfachen Dialogen und Teilaufgaben benutzt werden können.
- Es sollte stets der aktuelle, von Benutzer auch real angeforderte Inhalt in dem Moment präsentiert werden, in dem ihn der Benutzer auch benötigt.
- Es sollten dem Benutzer keine nicht nachvollziehbaren Wartezeiten zugemutet werden.
- Das System sollte einfach benutzbar sein, Fehler des Benutzers tolerieren, solange diese aufgelöst werden können und von hoher Benutzbarkeit sein.

Wir fassen diese vier Forderungen unter dem Stichwort HOME zusammen:

*H*igh quality content,
*O*ften updated,
*M*inimal download time und
*E*ase of use.

Die Dimensionen des Gestaltungsrahmens

Wir können den Gestaltungsrahmen um weitere Perspektiven erweitern und zugleich die obigen Be-

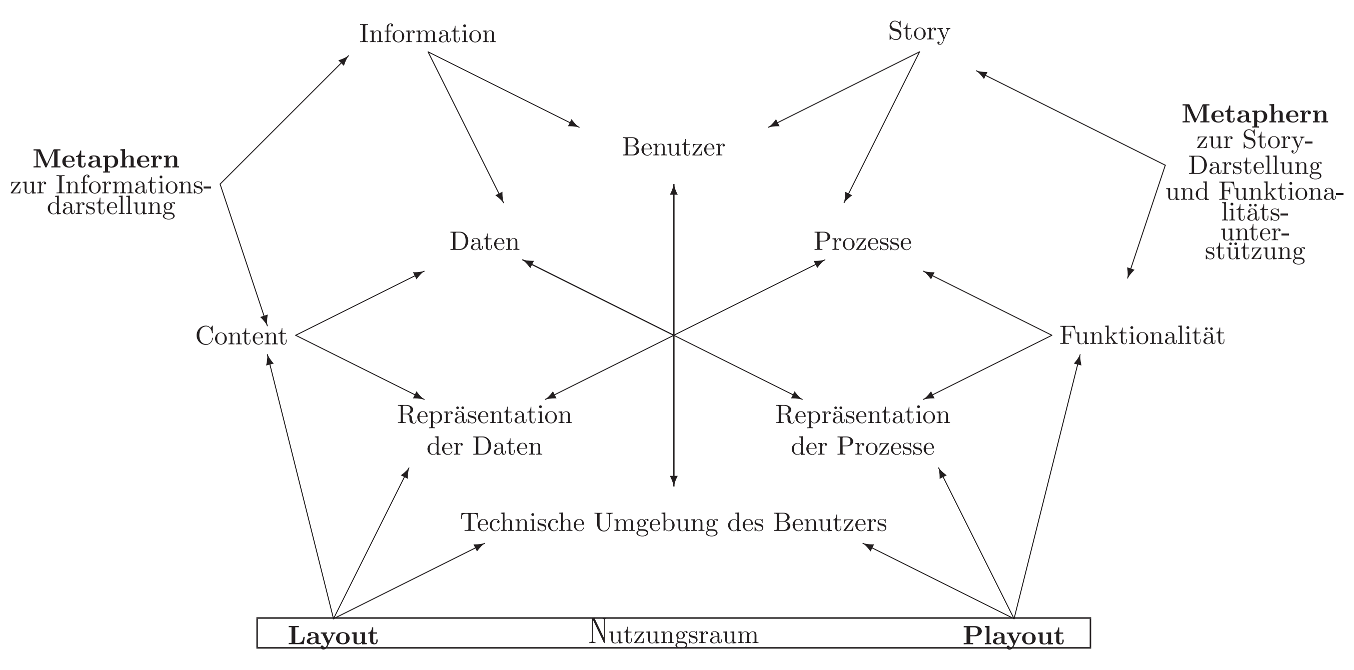


Bild 56: Dimensionen des Gestaltungsrahmens

In der Dimension des Benutzers wird der Benutzer entweder als Akteur charakterisiert oder mit seinem Profil und Portfolio angegeben. Einbezogen wird das Polaritätenprofil. Ableitbar ist dann die Zielgruppe und die erforderliche Anpassung.

In der Dimension der Daten werden die erforderlichen Sichten betrachtet.

In der Dimension der Datenrepräsentation werden Parameter zur Gestaltung der Oberflächen wirksam wie

- Form und Farbe,
- Kontrast und Rhythmus und
- Struktur und Komposition

eingesetzt.

In der Dimension der Prozesse werden die Abläufe der Story betrachtet.

In der Dimension des Prozeßrepräsentation werden die entsprechenden Implementationen der Stories dargestellt.

In der Dimension der technischen Umgebung des Benutzers werden die Potentiale der Verbindung, der Technik des Benutzers und der Server-Technik dargestellt. Diese Potentiale erlauben Effekte oder schränken diese stark ein.

Diese Gestaltungsdimensionen können auch in kombinierter Art und Weise zur Gestaltung herangezogen werden:

In den kombinierten Dimensionen Benutzer-Daten-Datenrepräsentation werden Metaphern zur Informationsdarstellung, mit denen ein Bezug auf die Benutzerwelten und die das Verständnis der Daten auf die Benutzer darstellen, eingesetzt.

In den kombinierten Dimensionen der Benutzer-Prozeß-Repräsentation werden Metaphern der Bewegung wirksam.

In der kombinierten Dimension der Daten-Benutzer-Welt wird der Informationsgehalt, der Wert der Information und die Benutzbarkeit der Information dargestellt.

In der kombinierten Dimension der Daten-Datenrepräsentation spielt die Bereitstellung der Daten als Content eine Rolle.

In der kombinierten Dimension der Prozesse-Prozeßrepräsentation wird aus der erforderlichen Funktionalität abgeleitet, welche gestalterischen Mittel erforderlich werden.

In der kombinierten Dimension der Prozeßrepräsentation-Umgebung wird das Playout abgeleitet. Es orientiert sich an

- dem Anliegen der Darstellung und setzt Anforderungen der Darstellung der Prozesse um,
- der Story selbst und
- den technischen Möglichkeiten, die durch die Umgebung gegeben sind.

In der kombinierten Dimension der Datenrepräsentation-Umgebung wird aus den zur Verfügung stehenden Informationen das Layout abgeleitet. Es orientiert sich an

- der Art des Content,
- dem Inhalt des Content und
- dem Anliegen der Benutzung
- sowie den technischen Möglichkeiten der Umgebung.

In der kombinierten Dimension der Benutzer-Prozesse wird die Story benutzt, um die Layout- und die Playout-Gestaltung abzuleiten.

Parallel zu den Dimensionen kann der Grad der Ausprägung bestimmt werden, mit dem Kategorien wie

- kraftvolle Gestaltung,
- angereicherte (wertebasierte) Gestaltung,
- erweiterte Gestaltung um Aspekte von
  - Verspieltheit (Romantik, Leidenschaft),
  - Kreativität mit einer Neuheit und überraschenden Effekten,
  - erfrischende Gestaltung mit Momenten einer Leichtigkeit und Transparenz,
  - Beruhigung durch Momente der Harmonie und der Ausgeglichenheit und
  - Anregung durch exotische und magische Elemente,
- natürliche Gestaltung mit einem Bezug auf das Umfeld des Benutzers und
- dynamische Gestaltung mit einer internen Bewegung und Spannung

benutzt werden, um eine Verstärkung oder Abschwächung zu erreichen.
Der Gestaltungsrahmen orientiert sich deshalb zentral auf

das **Layout** der Daten unter Berücksichtigung der technischen Möglichkeiten,

das **Playout** der Prozesse unter Berücksichtigung der technischen Umgebung des Benutzers,

Metaphern zur Informationsdarstellung und

Metaphern zur Story-Darstellung und Funktionsunterstützung

und mittelbar auf

die **Repräsentation der Daten** und

die **Repräsentation der Prozesse** wie z.B. die Orientierung im Nutzungsraum auf der Grundlage von

Feedback,

mentalen Modellen,

Metaphern zur Orientierung und

Gestaltungsrastern

und der Nutzungsdramaturgie der Interaktivität

durch Verfeinerung der Repräsentationen, d.h. durch Unterlegung der Repräsentation mit gestalterischen Mitteln.

Layout und Playout können zum **Nutzungsraum** zusammengefaßt werden.

Die Spezifikation des Gestaltungsrahmens wird somit durch folgende Spezifikation unterstützt:

- Beschreibung der Benutzer, der Akteure, der Benutzergruppen,
- Spezifikation der Story,
- Spezifikation der Prozesse,
- Spezifikation des Content,
- Spezifikation der Repräsentation der Daten,
- Spezifikation der Repräsentation der Prozesse und
- Angaben zur technischen Umgebung des Benutzers.

Der Gestaltungsrahmen legt die Gestaltungsgrundlagen fest. Dabei werden

- Prinzipien der visuellen Kollaboration,
- Prinzipien der visuellen Wahrnehmung und
- Prinzipien der visuellen Gestaltung

für die konkrete Aufgabe verfeinert.

Dazu werden Gestaltungslemente und Gestaltungsmittel eingesetzt.

Die Umsetzung des Gestaltungsrahmens

Wir erhalten eine Charakterisierung des Gestaltungsrahmens in tabellarischer Form:

Playout				Layout				Metapher		Akteure			Qualität
Funktion	Aufgabe	Darstellung	Kollaboration	Funktionen	Wahrnehmung	Kommunikation	Bildschirm	der Information	der Funktionalität	Zielgruppe	Polarität	Adaption	
...	...	...	...	...	...	...	...	...	...	...	...	...	...

In dieser Tabelle führen wir die Information zum Akteur zur direkten Assoziation mit. Sie dient eher der direkten Bezugnahme und bestimmt nicht den Gestaltungsrahmen. Die Qualitätsparameter dienen der Kontrolle.

Dieser Gestaltungsrahmen kann dann den Arbeitsoberflächen im speziellen oder dem Präsentationsraum im allgemeinen zugeordnet werden.

Wie bereits betont, kann dieser Gestaltungsrahmen verwendet werden, um Metaphern zu gewinnen. Die Spezifikation von Metaphern kann mit folgenden Metaphorikrahmen erfolgen:

Name der Metapher;

Die Darstellung der Eigenschaften assoziiert Eigenschaften der Metapher mit dem Anwendungsgebiet. Die Intensität und die Dominanz der Eigenschaften kann mit erfaßt werden.

Klasse der Metapher (personalisierte, allegorische, symbolische).

Bedeutung für unterschiedliche Benutzergruppen in unterschiedlichen kulturellen Kontexten.

Repräsentation der Metapher durch entsprechende Formen und Farben, Kontrast und Rhythmus, Struktur und Komposition.

Die Zuordnung von Metaphern zum Gestaltungsrahmen und zu den Content-Suiten sowie zur Funktionalität erfolgt explizit

durch Angabe der Suiten (Content-Suiten, Funktionen, Container, Akteure),

durch Angabe der Metapher-Einbettung mit den Parametern für

- die Funktion der Metapher in der Suite,
- die Anwendbarkeit der Metapher im Gestaltungsrahmen z.B. als Vollseiten-, Teilseiten-, Beiwerk-Metapher,
- das Ursprungsgebiet der Metapher,
- die Abstraktionsrichtung (Generalisierung/Spezialisierung),
- die Richtung (vom Lebewesen zum Gegenstand, vom Gegenstand zum Lebewesen),
- dem beabsichtigten Effekt (prädikativ oder überzeugend; attributiv, genitiv, kompositional, Apposition) und
- die Repräsentationsform als Auswahl unter den verschiedenen möglichen Repräsentationsformen der Metapher, sowie

durch Angabe der Intention der Metapher mit Parametern für

- den Kontext (Intention für Akteur, Erwartungen des Akteurs, Co-Notationen (soziale und emotionale)),
- der Funktion (intern, prädikativ, heuristisch, emotional, sozial, rhetorisch, ästhetisch) und
- dem Typus der Metapher (konventionell, kreativ, Ex-Metapher, Re-Metapher).

Wir können wiederum anstelle einer Tabelle Arbeitsblätter zur Darstellung der Metaphern verwenden.

Der Gestaltungsrahmen wird außerdem durch eine Spezifikation der

Betonung,

Dominanz,

Transparenz und der

Kontrastierung

ergänzt. Mit dieser Darstellung können wir auch die Akzeptanz, die Wahrnehmung, die Aufnahme des Inhaltes, seine Verarbeitung und die Stimmung beeinflussen.

Diese Spezifikation kann auch als Interface-Polaritätenprofil mit einer ordinalen Bewertungsskala zur Darstellung der Ausgeprägtheit der Eigenschaften erstellt werden.

Perspektiven der Mensch-Computer-Interaktion

Traditionell werden vier verschiedene Perspektiven der Interaktion unterschieden:

Maschinenperspektive: Der Computer wird bei den Betrachtungen in den Mittelpunkt gestellt. Der Mensch wird als Maschinenbediener wahrgenommen.

Systemperspektive: Mensch und Computer werden als gleichberechtigte Partner eines Systemes aufgefaßt.

Kommunikationsperspektive: Programme und Benutzer unterhalten sich gleichberechtigt in einer Dialogsprache.

Werkzeugperspektive: Der Computer unterstützt den Benutzer bei der Erfüllung seiner Aufgaben.

Auf der Grundlage dieser Perspektiven können wir vier Methoden zur Entwicklung von Oberflächen ableiten:

Die empirische Methode orientiert sich an den aktuellen Herangehensweisen, verallgemeinert diese und gestattet die Entwicklung in einem trial-and-error-Prozeß.

Die kognitive Methode untersucht zuerst das Verhalten des Menschen, ergründet sein mentales Modell und nutzt diese Erkenntnisse zur Entwicklung von benutzungsfreundlichen und intuitiv verständlichen Oberflächen.

Die prediktive (funktionsorientierte) Methode leitet aus den Zielen der Anwendung, den Aufgaben und den technischen Möglichkeiten eine Lösung für die Gestaltung des Arbeitsprozesses und der entsprechenden Oberflächen ab.

Die anthropomorphe Methode bildet die Kommunikation von Mensch und Maschine in Form von Dialogen, Dialogobjekten und Oberflächen nach.

Meist wird eine Kombination dieser Methoden gewählt. Eine bevorzugte Variante ist die Kombination der prediktiven und der anthropomorphen Methoden.

Aus diesen Perspektiven sind zwei grundsätzliche Zugänge zur Gestaltung von Oberflächen ableitbar:

Der konstruktive Ingenieurzugang orientiert sich an den Entwicklern und den vorhandenen technischen Möglichkeiten und damit an der Maschinenperspektive. Systeme dieser Bauart können einfach und elegant, einfacher auch durch den Benutzer zu pflegen und für den eingearbeiteten Benutzer auch durchschaubar sein.

Der Benutzer-Aufgaben-Zugang beruht auf eine Kombination der Werkzeug-, Kommunikations- und der Systemperspektive. Ausgehend von einem Aufgabenmodell und einem Interaktionsmodell wird der Computer zum Partner bei der Lösung einer Arbeitsaufgabe. Die einzelnen Arbeitsschritte werden in Oberflächen nachgebildet. Der Vorteil dieses Zugangs ist die leichte Erlernbarkeit. Von Nachteil ist die Fixierung auf den aktuellen Zustand des Arbeitsprozesses, der u.U. auch von bislang benutzten, nicht adäquaten Werkzeugen und deren Funktionalität geprägt ist.

8 Sprachen zur Darstellung der Verteilung

Entzwei' und gebiete! Tüchtig Wort;
Verein' und leite! Bessrer Hort.
Goethe, Sprichwörtlich

Allgemeine Architektur von Kollaborations- und Diensteverwaltungssystemen
Wir modellieren die Verteilung von Informationssystemen als Kollaboration oder *Zusammenarbeit* von Systemen. Systeme oder Akteure arbeiten temporär zur gemeinschaftlichen Lösung von Aufgaben zusammen. Sie bilden einen temporären Verbund oder eine temporäre Arbeitsgruppe, verfügen über gemeinsame Arbeitsinstrumente, z.B. eine Objekt-Suite, und folgen einem Kollaborationsvertrag. Die Kollaboration hat dabei drei Facetten:

Koordination: Das ordnende Zusammenfassen, die Abstimmung und die Zuordnung verschiedener Arbeitsaufgaben wird durch einen *Koordinationsrahmen* gewährleistet. Koordination bezeichnet also jene Teile der Kollaboration, die zur Abstimmung aufgabenbezogener Tätigkeiten notwendig sind.

Kommunikation: Die Abstimmung der Partner in einer Kollaboration wird durch einen *Nachrichtenaustausch* zwischen den Partner mit einem entsprechenden *Austauschrahmen* realisiert.

Kooperation ist die *Tätigkeit mehrerer Partner* zur Verwirklichung eines Zieles, bei der jeder Partner bestimmte Teilaufgaben übernimmt, diese dem Partner gegenüber abrechnet und bei Nichterfüllung der Verpflichtung eine kompensierende Teilaufgabe auslöst. Kooperation bezeichnet also jene Teile der Kollaboration, die zur Koordination und zur Vereinbarung gemeinsamer Ziele notwendig sind.

Diese unterschiedlichen Blickwinkel müssen bei einer Modellierung der Verteilung mit unterlegt werden. Deshalb benutzen wir das Kollaborationsdienstemodell in Bild 57. Dieses Modell verallgemeinert und erweitert das Dienstemodell in Bild 13.

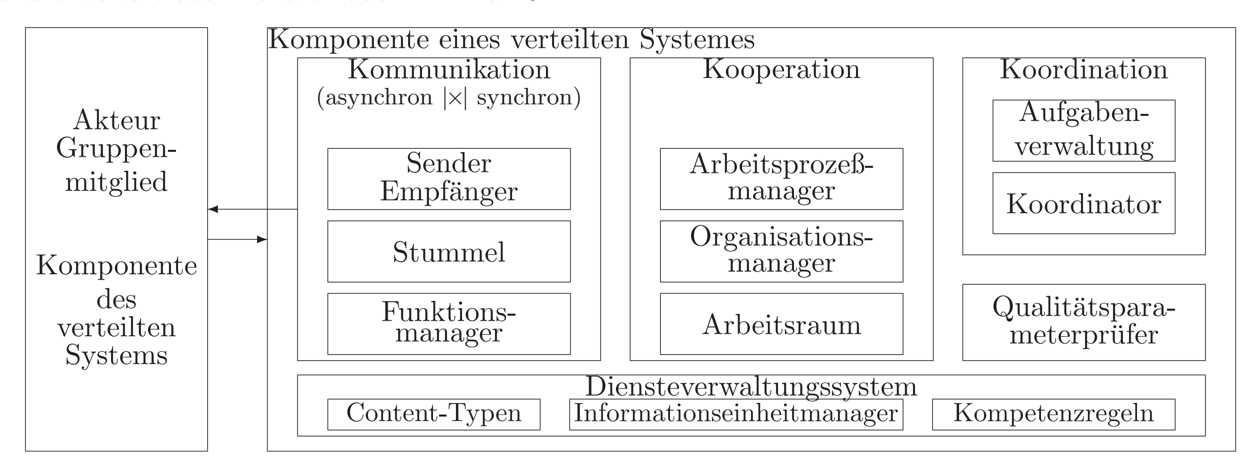


Bild 57: Die Implementationsschicht eines verteilten Systems

Die Verteilung unterstützt die Kollaboration. Die übliche Form ist die Kollaboration von Systemen. Eine spezifische Form der Kollaboration ist die Zusammenarbeit von Gruppen (CSCW - computer supported cooperative work). Gruppenarbeit ist die Summe aller aufgabenbezogenen Tätigkeiten, die von Gruppenmitgliedern ausgeführt werden, um zielbezogene Aufgaben zu erfüllen und somit das Gruppenziel zu erreichen.

Der Kollaborationsdienst wird im Rahmen der Implementierung durch einen konkreten *Kollaborationsrahmen* unterlegt. Dieser Kollaborationsrahmen setzt auf einer verfügbaren Kollaborationsarchitektur, einem Kollaborationsstil und einem Kollaborationsmuster auf.

Das Abstraktionsschichtenmodell erlaubt eine Separation der Spezifikation in

Anwendungsdienst mit einer Darstellung des Verhaltens auf Anwendungsebene

abstrakter Dienst der Aktionsschicht mit einer Darstellung der Kollaboration,

konzeptioneller Dienst mit einer Darstellung des Kollaborations- und Dienstesystemes auf konzeptioneller Ebene und

Systemdienst mit einer Angabe der Systemroutinen, - programme und Protokolle.

Diese Dienste können wie in Bild 58 geschichtet werden.

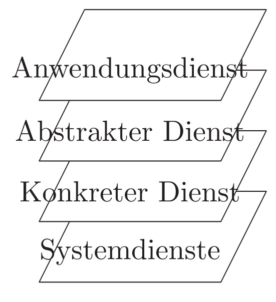


Bild 58: Eine Schichten-Architektur für verteilte System

C. J. Date stellte 12 'Regeln' für verteilte DBS auf:

1. Größtmögliche lokale Autonomie und lokale Verwaltung von lokalen Daten;
2. Keine Abhängigkeit vom zentralen Knoten;
3. Permanenter Betrieb;
4. Ortsunabhängigkeit (Ortstransparenz), d. h. die physische Lokation von Daten muß verborgen bleiben und Datenumverteilungen dürfen keine Auswirkungen auf Programme haben;
5. Partitionierungsunabhängigkeit;
6. Replikationsunabhängigkeit;
7. Verteilte Anfrage-Bearbeitung, die für den Zugriff auf externe Daten und die Optimierung verteilter Anfragen erforderlich ist;
8. Verteilte Transaktionsverwaltung, einschließlich Synchronisation, Recovery (verteiltes Commit-Protokoll);
9. Hardware-Unabhängigkeit;
10. Betriebssystemunabhängigkeit;
11. Netzwerkunabhängigkeit;
12. DBMS-Unabhängigkeit.

Nicht jedes dieser Kriterien wird durch die kommerziellen Systeme befriedigt, z. B. ist das Kriterium 10 bei einigen Firmen im Interesse der Firmenpolitik nie unterstützt worden. Die meisten dieser Regeln führen direkt zu heterogenen DBMS.

Die meisten kommerziellen DBS unterstützen eine Teilfunktionalität von verteilten DBS. Beispiele kommerzieller Systeme sind Tandem NonStop SQL, CA Ingres/Star; CA-DB:STAR, Oracle, Informix/Star, Sybase Replication Server, IBM DRDA (DB2, DB2/2, DB2/6000, SQL/DS, ...), Cincom Supra, Empress, UDS-D und Sesam-DCN. Frühe Prototypen sind z. B. die Systeme R* (IBM), SDD-1 (CCA), Distributed Ingres, VDN, POREL, DDM, DDTS, Sirius-Delta und Polypheme.

Facetten der Verteilungsqualität

Die Qualität der Verteilung wird aus unterschiedlichen Gesichtswinkeln beurteilt. Die Bandbreite der Anforderungen ist sehr hoch. Wir können dies an einer Reihe von Dimensionen feststellen:

Allgegenwart (Ubiquity): Daten haben je nach Anforderungen eine unterschiedliche Aktualität. Typische entgegengesetzte Anwendungen sind

nomadische Informationssysteme, bei denen Datenbestände der einzelnen Knoten ggf. auch zu einem späteren Zeitpunkt aktualisiert und abgeglichen werden, und

fest miteinander kooperierende Informationssysteme, in denen Kopien von Objekt-Suiten über Replikationsmechanismen miteinander verbunden sind und die alle gleichzeitig die gleiche Aktualität haben.

Mobilität der Rechner bedeutet zugleich auch eine Herausforderung an die derzeitige Technologie und bedarf einer Unterlegung durch entsprechende Spezifikationsmethoden, wobei im wesentlichen vier Aspekte genauer untersetzt werden müssen:

Portabilität: Da zu unterschiedlichen Zeiten unterschiedliche Plattformen verwendet werden, kann durch eine entsprechende Portabilität ein Gleichklang der Anwendung erreicht werden.

Freizügigkeit und Offenheit: Das verteilte System ist erweiter- und reduzierbar ohne Einbußen der Qualität hinnehmen zu müssen. Die Schnittstellen enthalten keine verborgene Funktionalität und sind voll veröffentlicht.

Erreichbarkeit: Anwendungen sollen zu jeder Zeit, von jedem Ort, durch jeden zugelassenen Benutzer zu den besten Bedingungen mit adäquatem Content erreichbar sein.

Beweglichkeit: Ein Benutzer kann sich mit seiner Plattform innerhalb eines Netzes fortbewegen ohne Einbußen bei der Dienstqualität hinnehmen zu müssen. Einem Benutzer ist das Andocken an einen Dienst ohne Einschränkungen durch den Dienst erlaubt.

Sicherheit: Klassische Sicherheitanforderungen sind

- *Vertraulichkeit* (Schutz gegen Offenlegung gegenüber nicht berechtigten Benutzern, Akteuren oder Systemen),
- *Integrität* (Schutz gegen Veränderung oder Beschädigung) und
- *Verfügbarkeit* (Schutz gegen Störungen der Bearbeitung).

Neben diesen Anforderungen treten in verteilten Anwendungen aufgrund der *Unsicherheit* des Kommunikationsmediums zwei weitere:

- Ein (maskierter) Benutzer kann einen Dienst durch eine Vielzahl von Anforderungen ersticken. Durch explizite Spezifikation der *Verweigerung von Diensteangriffen* wird dem entgegengewirkt.
- Ein (maskierter) Benutzer kann Würmer, trojanische Pferde oder Viren mit dem Ziel versenden, sich (unberechtigten) Zugang zu den Daten oder Leistungen des Dienstes zu verschaffen. Deshalb ist eine *Sicherung des mobiles Codes und der mobilen Daten* erforderlich.

Bedeutungstreue erfordert die Integration der Benutzerwelt in die Spezifikation. Sie umfaßt unterschiedliche Aspekte:

Benutzerverständnis: Das Verständnis der Benutzer über den Content ist frei von Konflikten. Unterschiedliche Content-Suiten besitzen eine verschiedene Bedeutung.

Temporale Aspekte: Content-Suiten entwickeln sich über die Benutzungszeiträume. Die Zeitaspekte können konzeptionell mit den Content-Suiten verbunden werden.

Context: Content-Suiten besitzen einen Kontext, der sie nicht in Konflikt miteinander bringt.

Bedeutungstreue kann durch eine Theorien- und Modellierungswelt und eine abgestimmte Begriffslandkarte unterstützt werden. Die Entwicklung einer Spezifikationsmethode zur Unterstützung der Bedeutungstreue ist jedoch noch ein Forschungsthema.

Heterogenität ist in verteilten Systemen die Regel. Sie tritt in unterschiedlichen Facetten auf:

Heterogenität der Netzwerke,

Heterogenität der Plattformen verursacht durch unterschiedliche

Betriebssysteme,

Hardware,

Datenbank-Management-Systeme bzw.

Programmiersprachen,

Heterogenität durch unterschiedliche Programmiermethodiken,

Heterogenität der Datenbankschemata,

Heterogenität der Funktionalität der Informationssysteme und

Heterogenität der Story-Räume.

Nicht alle Spielarten der Heterogenität können durch entsprechende Kollaborationssysteme überwunden werden. Hauptmechanismen zur Überwindung der Heterogenität sind entsprechende Abbildungen bzw. Mediatoren.

Konsistenz: Objekt-Suiten dürfen keine Konflikte der unterschiedlichen Komponenten der Kollaboration untereinander besitzen. Die Konsistenz wird durch die Lösung einer Reihe von Problemen unterstützt:

Herausfiltern potentieller Konflikte: Die Konsistenz von Objekt-Suiten kann berechnet werden, solange Objekt-Suiten hierarchisch strukturiert sind und die Integritätsbedingungen stratifizierbar sind. In vielen Fällen ist ein Herausfiltern von Konflikten möglich. Das Auffinden von Inkonsistenzen kann durch die Modellierung der Inkonsistenz- und Konfliktbedingungen einfacher werden.

Beweis von Eigenschaften: Der Nachweis von Spezifikationskonflikten ist an die Axiomatisierbarkeit bzw. Entscheidbarkeit von Klassen von Integritätsbedingungen gebunden. Deshalb können nur einfache Eigenschaften geprüft werden.

Pflege der Konsistenz bei Modifikation: Eine Modifikation der Datenbank kann auch die Konsistenz zerstören. Mit Transaktionsmodellen kann dem entgegengewirkt werden.

Konsistenzinseln erlauben auch die Benutzung von global inkonsistenten Datenbanken. Sie müssen explizit formuliert werden.

Dauerhaftigkeit: Die Persistenz von Veränderungen erfordert eine Abspeicherungsstrategie für erfolgte Veränderungen bzw. eine Zurückführungsstrategie. Die Dauerhaftigkeit wird durch die klassische Datenbanktechnologie unterstützt.

Robustheit von Systemen erfordert Fehlertoleranz. Es treten unterschiedliche Arten von Fehlern auf:

Benutzerfehler werden durch eine falsche Benutzung der Dienste durch autorisierte Benutzer verursacht. Durch eine gute Gestaltung des Story-Raumes können diese Fehler vermieden werden. Ihre Behebung ist deshalb nicht Teil der Spezifikationsanforderungen für verteilte Systeme.

Systemfehler können in verteilten Systemen in unterschiedlicher Form auftreten:

- *Diensteverweigerungs- und Auslassungsfehler* entstehen durch Nichtausführung eines Befehles durch den Kollaborationsdienst oder den Kommunikationskanal. Da ein allgemeines Fehlererkennungsmodell nicht existieren kann, ist eine Spezifikation zur Behebung möglicher Fehler der einzige Ausweg.

- *Zufällige Fehler* oder byzantinische Fehler sind alle Fehler, die nicht eindeutig zugeordnet werden können bzw. die eine Beliebigkeit aufweisen.
- *Timing-Fehler* treten beim Vorhandensein von Zeitschranken oder bei Synchronisierung auf. Ein Prozeß kann nicht rechtzeitig ausgeführt oder ein Resultat kann nicht rechtzeitig übertragen worden sein. Außerdem kann der Abstimmungsprozeß oder die Zeitmessung fehlerhaft sein.

Skalierbarkeit: Ein verteiltes System soll invariant gegenüber Veränderungen in der Anzahl der Ressourcen und Benutzer sein. Skalierbarkeit erfordert die Lösung einer Reihe von Problemen, wie z.B. die folgenden:

Kostenkontrolle der physischen Ressourcen: Falls das verteilte System sein Leistungsspektrum ausschöpft, sollte eine Erweiterung möglich sein, ohne eine Re-Modellierung, Re-Implementation oder Austausch der Infrastruktur vornehmen zu müssen. Wird eine Erweiterung notwendig, dann kann diese Erweiterung ohne Kostenexplosion realisiert werden.

Kontrolle des Leistungsverlusts: Mit der Erweiterung des Systems soll der Anstieg der Kosten kontrollier- und abschätzbar sein. Durch eine entsprechende Architektur des Systems, wie z.B. hierarchische Strukturierung, kann der Leistungsverlust minimiert werden.

Verhinderung des Ausschöpfens der Ressourcen: Ressourcen, wie Adreß- und Namensräume, sollten sich beliebig erweitern lassen, sobald der Bedarf ansteigt.

Vermeidung von Leistungsengpässen: Sind einige Komponenten eines verteilten Systems dauerhaft an der Leistungsgrenze, dann muß eine effektive und einfache Abhilfe möglich sein. Leistungsengpässe kann man durch Replikation und kontrollierte Redundanz partiell überwinden.

Transparenz wird in der Informatik[15] als das Verbergen der einzelnen Komponenten vom Benutzer des verteilten Systems verstanden. Wir unterscheiden unterschiedliche Formen der Transparenz:

Zugriffstransparenz unterstützt den Zugriff auf Dienste unabhängig von deren Ort mit den gleichen Funktionen.

Ortstransparenz unterstützt den Zugriff trotz der Unkenntnis vom Ort des Dienstes.

Nebenläufigkeitstransparenz erlaubt mehrere Prozesse gleichzeitig zu fahren, ohne daß sich diese gegenseitig beeinflussen.

Replikationstransparenz erlaubt die Benutzung von Kopien, ohne daß ein Benutzer dies wahrnimmt.

Fehlertransparenz wird durch das Verbergen von Fehlern vor dem Benutzer unterstützt.

Mobilitätstransparenz erlaubt das Verschieben von Diensten und insbesondere von Ressourcen.

Leistungstransparenz erlaubt eine Selbstreorganisation des Systems ohne Beeinträchtigung des Benutzers.

Skalierungstransparenz erlaubt eine Veränderung der Systemgröße ohne Beeinträchtigung der anderen Funktionen.

Verträge zur Verteilung

Der Vertragsraum besteht aus einer Darstellung der Dienste, aus dem Kollaborationsvertrag und dem Qualitätsvertrag. Im einzelnen werden die folgenden Elemente dargestellt:

Das Dienstmodell (*“was”*) basiert auf einer allgemeinen Darstellung der Dienste mit den Sichtenskizzen und den Kompetenzen der Dienste.

[15] “Transparenz” ist im deutschen Sprachgebrauch im Sinne von (Licht-)Durchlässigkeit

Der Kollaborationsvertrag (*"wie"*) stellt dar,

- welche Partner ("wer", "mit wem"),
- welche Dienstekomponenten ("was"),
- mit welcher Qualität,
- mit welchen Verantwortlichkeiten ("woraus"),
- auf welcher allgemeinen Vertragsgrundlage ("worauf") und
- mit welchem Workflow ("wie")

benutzen oder bereitstellen.

Das Qualitätsmodell (*"in welcher Qualität"*) stellt die allgemeinen Qualitätsparameter zusammen.

Das Zeitmodell (*"wann"*) stellt analog zu Ablaufmodellen den Verlauf der Kollaboration dar.

Das Kontextmodell (*"warum"*) stellt den Kontextrahmen dar.

Das Akteurmodell (*"wer"*) legt die Partner mit ihren Rollen und Rechten fest.

Der Kollaborationsvertrag dient der Sicherung der Qualität des Dienstes. Er regelt die Herangehensweise bei einer Vertragsverletzung, die Verantwortlichkeiten und die Rollen der einzelnen Kollaborationspartner. Er kann ggf. durch eine zentrale Konfliktbehebung unterstützt werden. Wir verwenden zur Sicherung des Kollaborationsvertrages eine Systemkomponente, den Qualitätsparameterprüfer.

Der Qualitätsparameterprüfer basiert auf zwei zentralen Modellen:

Fehlermodell: Es existieren eine Reihe von Techniken zur Bearbeitung von Fehlern:

Erkennung von Fehlern: Damit Fehler erkannt werden können, müssen gesonderte Prüfmechanismen den Spezifikationen hinzugefügt werden. Typische Prüfmechanismen sind Prüfsummen der Kodierungstheorie und Hilfssichten, die zur Modifikationsfähigkeit von Sichten bei nicht modifizierbaren Sichten verwendet werden.

Maskierung von Fehlern: Fehler können durch erhöhte Funktionsredundanz (wiederholte Ausführung oder Übertragung) oder Datenredundanz (RAID-Speicherung) abgeschwächt werden.

Tolerierung von Fehlern: Die Akzeptanz von Fehlern durch die Benutzer oder durch die Systeme ist ein Ausweg aus dem Entstehen von Fehlern.

Wiederherstellung nach Fehlern: Die Wiederherstellung eines korrekten Systemzustandes ist für DBMS hinreichend gelöst.

Der Hauptmechanismus zur Behebung von Fehlern sind Mehrphasen-Commit-Protokolle und kompensierende Transaktionen, die bereits als Elemente der Funktionalität von Informationssystemen vorgestellt wurden.

Sicherheitsmodell: Die Sicherheit kann durch entsprechende Datenbanktechniken unterstützt werden:

Sicherungssichten: Die Benutzer arbeiten grundsätzlich nur auf Sichten, nicht aber auf den Basisdaten.

Integritätspflege: Die Integritätsbedingungen werden zusammen mit den Pflegestrategien spezifiziert und durch entsprechende Prozeduren unterstützt. So kann z.B. eine Zusammenfassung von Funktionen in stored procedures eine Pflege der Integrität unterstützen.

Datenbankmonitor: Es wird durch einen Monitor ein Abzug des Systemzustandes visualisiert. Gleichzeitg werden entsprechende Eingriffsroutinen bereitgestellt.

Diese Verfahren können mit expliziten Abwehrmaßnahmen und kryptographischen Verfahren

Diese Modelle werden zu Modellen ergänzt, mit denen andere Qualitätsparameter garantiert werden können. Der Qualitätsparameterprüfer kann diese Modelle ggf. auch unterstützen.

Innerhalb der Aktionsschicht wird *Allgegenwart* unterstützt. Dazu sind systemtechnisch entsprechende Voraussetzungen zu schaffen.

Innerhalb der konzeptionellen Schicht betrachten wird zwei weitere Qualitätsparameter:

- Bedeutungstreue wird durch eine Einbindung von Konzepten und Begriffen unterstützt.
- Konsistenz wird mit dem Integritätsmanagement verknüpft.

Die Qualitätsparameter der konzeptionellen Schicht werden auf entsprechende Funktionen der Implementationsschicht durch Instantiierung des Austauschrahmens abgebildet. Außerdem können je nach Bedarf folgende Qualitätsparameter unterstützt werden.

- Dauerhaftigkeit,
- Performanz,
- Robustheit,
- Skalierbarkeit und
- Transparenz.

Der Kollaborationsraum

Bislang sind formale Methoden zur Spezifikation der Kollaboration nicht entwickelt worden. Auf der Grundlage der vorangegangenen Überlegungen sind wir jedoch in der Lage eine allgemeine Theorie des Kollaborationsraumes zu entwickeln. Kollaboration spielt sich in den drei Facetten

- Kommunikation,
- Kooperation und
- Koordination

ab.

Der Kollaborationsrahmen von Seite 134

Kollaborationsrahmen													
Kollaboration					Art des Zusammenwirkens			Arbeitsplatz					
Form	Rollen	Formierung	Raum / Zeit	Vertrag	Beziehungen	Arten	Diskurstyp	Akteure	Gruppe	Rechte	Portfolio	Organisation
...	...	...	...	...	...	...	...	...	...	...	...	...

wird nun aufgespleißt in drei Kollaborationsrahmen, die unterschiedliche Facetten der Kollaboration, des Zusammenwirkens und des Arbeitsplatzes darstellen. Wir können dieses Aufspleißen in einer Tabelle oder wie auf Seite 157 durch ein Arbeitsblatt darstellen. Da die Darstellung als Arbeitsblatt kondensiert ist, wenden wir uns zuerst der Darstellung durch Tabellen zu und untersuchen die Facetten der Kollaboration einzeln. Mit der Vereinheitlichung und den einzelnen Kollaborationsrahmen erkennen wir außerdem, welche Aspekte eine genauere Untersetzung bei der Spezifikation erfordern.

Der Kommunikationsrahmen stellt die Austauschformen zur Verfügung. Wir können hierbei auf der ORB-Architektur aufsetzen. Durch die Object Management Group (OMG) wurde die in Bild 59 und Bild 60 dargestellte Object Management Architecture (OMA) verabschiedet. Sie gestattet eine höhere Interoperabilität durch standardisierte Zugriffsschnittstellen. Die Schnittstellenbeschreibung erfolgt durch IDL (Interface Definition Language). Der Object Request Broker ist der Vermittler in der Client-/Server-Kooperation zwischen Objekten. Ein Aufruf besteht aus dem Tripel (Operationsname, Zielobjekt, Parameter). Damit wird eine Ortstransparenz realisiert. Die Objektdienste (Object

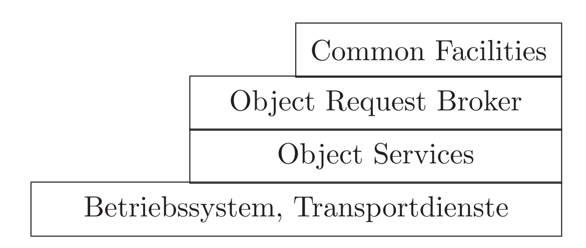


Bild 59: CORBA auf IDL Grundlage

Services) realisieren Basisfunktionen für die Erzeugung und Verwaltung von Klassen und Objekten, zur Namensverwaltung und für die Persistenz von Datenbank-Objekten. Mit den Common Facilities werden allgemeine Hilfsfunktionen (Klassenbibliotheken) zur Verfügung gestellt.

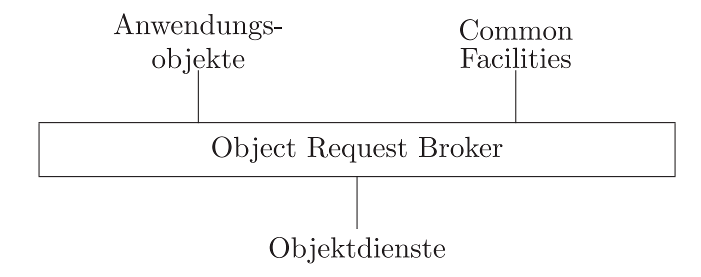


Bild 60: OMG - Architektur

In der Realisierung von OMA in der Common Object Request Broker Architecture (CORBA) in Bild 61 sind statische und dynamische Methodenaktivierungen (Aufrufschnittstellen) realisiert. Die ORB-Schnittstelle ermöglicht einen Zugriff auf Infrastrukturfunktionen, z. B. für die Verwaltung globaler OIDs und die Registrierung von Objekten. Die Kommunikation zwischen ORBs wird über das IIOP (Internet Inter-ORB-Protokoll) realisiert.

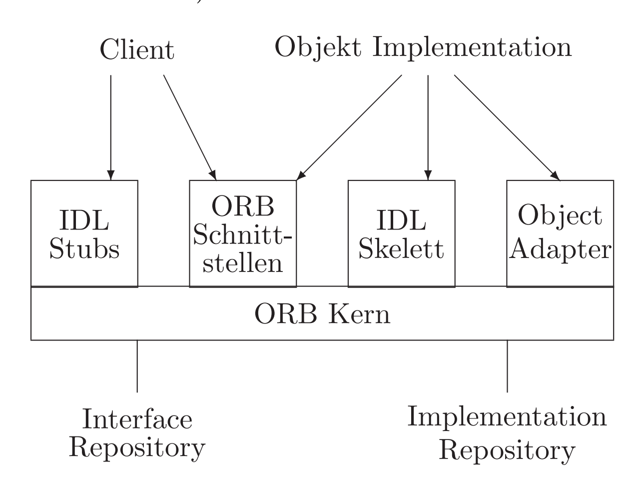


Bild 61: CORBA - Architektur

Damit erhalten wir eine Spezialisierung des Kollaborationsrahmens zu einem Kommunikationsrahmen:

Kommunikation					Funktionsmanager			Stummel				
Form	Synchronisation	Formierung	Raum / Zeit	Sicherheit	Regeln	Funktionen	Protokolldiskurs	Partner	Gruppe	Rechte	Portfolio	Protokollablauf

Kooperation basiert auf kooperierenden Sichten. Es werden dazu koordinierende Formeln angegeben, die Typen eines Schemas mit Typen eines anderen Schemas assoziieren.

Zur Spezifikation der Kooperation werden die drei Bestandteile dargestellt:

- Mit der *Kooperationsform* wird die Art der Kooperation dargestellt. Es wird das Zustandekommen einer Kooperation durch die gewählte Form, durch die Geschichte des Zustandekommens, die Parallisierbarkeit und den gewählten Vertrag charakterisiert.

- Die Kooperation unterliegt einem Ablauf, den wir im *Arbeitsprozeß der Kooperation* zusammenfassen. Es werden die Organisation der Kooperation, die Partner und der Ablauf der Kooperation beschrieben.

- Die Kooperation wird durch Dienste unterstützt. Wir fassen die Dienste, deren Nutzung, Bereitstellung und Veränderung im *Austauschrahmen* zusammen. Die einzelnen Partner haben innerhalb von Gruppen spezifische Arbeitsaufgaben, für die Dienste zur Verfügung gestellt werden, wobei die Organisation der Kooperation mit erfaßt wird.

Wir erhalten damit den folgenden Kooperationsrahmen:

Kooperationsform					Arbeitsprozeßverwaltung			Austauschrahmen				
Form	Rolle	Formierung	Parallelität	Vertrag	Arbeitsprozeß	Partner	Diskurstyp	Partner	Gruppe	Dienste	Portfolio	Organisation
...	...	...	...	...	...	...	...	...	...	...	...	...

Die Koordination ist der dritte Bestandteil der Kollaboration. Es gibt relativ wenig Literaturquellen zur Darstellung der Koordination. Im Rahmen unserer Sprache zur Spezifikation der Verteilung DistLang streben wir eine umfassende Spezifikation der Kollaboration an, so daß wir auf eine Spezifikation der Koordination nicht verzichten wollen. Es werden nicht nur die Zeit- und Vertragsbeschränkungen abgebildet, sondern vor allem die Art des Zusammenwirkens spezifiziert.

Eine Spezifikation der Koordination umfaß daher die folgenden Elemente:

Durch die Koordinationsform werden die Form der Koordination, die Rollen in der Koordination, die Vor- und Nachbedingungen für die Formierung, die Abstimmung bzw. Synchronisation der Partner und das Fehlermodell zusammengefaßt.

Die Aufgabenverwaltung basiert auf einer Spezifikation der Aufgaben, der Zuordnung der Aufgaben zu den Partnern und dem Verlauf der Koordination.

Der Koordinator stellt die einzelnen Partner, die Rolle der Partner in einer Gruppe, die verwendete Infrastruktur (Dienste), die Aufgaben und die Abbildung der verwendeten Namen in einer integrierten Form zusammen.

Als Beispiel kann das koordinierte Zusammenwirken beim Erstellen eines Vorschlages für eine Lehrveranstaltung betrachtet werden. Mitglieder eines Lehrstuhles wirken koordiniert zusammen, indem ein Mitglied die Rolle des Koordinators übernimmt und die einzelnen Mitglieder schrittweise ihre Obligationen erfüllen.

Ein anderes Beispiel der Koordination ist das Zusammenwirken eines Programmkomitees. Dieses Komitee wird durch einen virtuellen Koordinator zur Erfüllung von Aufgaben (Erstellung von Gutachten zu Arbeiten z.B.) angehalten, ggf. auch durch entsprechende Nachrichten ermahnt und durch Nachrichten über den Fortschritt der Kollaboration informiert.

Wir können deshalb den folgenden *Koordinationsrahmen* zur Spezifikation verwenden:

Koordinationsform					Aufgabenverwaltung			Koordinator				
Form	Rolle	Formierung	Synchronisation	Fehlermodell	Aufgabe	Zuordnung	Diskurstyp	Partner	Gruppe	Dienste	Portfolio	Namensraum

Die unterschiedlichen Bestandteile von Kommunikation, Kooperation und Koordination können für das Arbeitsblatt zum Austauschrahmen zusammengefaßt werden, der aus drei Dimensionen besteht:

Architektur: Die Architektur stellt einen Zusammenhang der verwendeten Dienste und Komponenten her.

Kollaborationsstil: Der Kollaborationsstil stellt die Modelle zur Unterstützung, Zugriff, zur Implementation der Kollaboration und den Kollaborationsdiskurs dar.

Kollaborationsform: In der Kolloaborationsform werden die Konfigurations- und Zugriffsparameter, die Ereignisverarbeitung, die Synchronisation und die Parameter des Kollaborationsvertrages untersetzt.

Eine alternative Darstellung wird durch eine Darstellung von

Architektur,

Formation und

Berechnung

wie im Arbeitsblatt auf Seite 157 gegeben.

Spezifikation auf der Implementationsschicht: Das Dienste- und Kollaborationsverwaltungssystem

Das Dienste- und Dienste- und Kollaborationsverwaltungssystem wird mit einer Dienstnehmer-Dienstgeber-Architektur in die Infrastruktur eingepaßt. Dazu unterscheiden wir zwischen Datenübertragung und Datenverwaltung für jede Komponente. Zur allgemeinen Darstellung von verteilten Systemen für die Implementationsschicht verweisen wir auf Monographien und Lehrbücher zu verteilten Systemen wie z.B. [ALSS03].

Produkte des Spezifikationsprozesses für die Verteilung

Produkte der Spezifikation der Verteilung sind je nach Abstraktionsschicht:

Im ***Pflichtenheft*** werden die Dienste, der Kollaborations- und der Qualitätsvertrag festgehalten.

Die Spezifikation des ***kollaborativen Systemes*** umfaßt zur Spezifikation des Kollaborationsraumes den Kollaborationsrahmen, die Dienste aus der Benutzersicht und die Qualitätsparameter, die ein Benutzer beurteilen kann.

Das ***Dienste- und Kollaborationssystem*** spezifiziert den Diensteraum mit den Content-Typen, den Kollaborationsrahmen und die Architektur des Dienste- und Kollaborationssystemes.

Das ***Dienste- und Kollaborationsverwaltungssystem*** ist beschrieben durch die Implementation des verteilten Systemes. Es werden insbesondere die Protokolle, die Verteilung, die unterstützenden Programme und die Qualitätsverwaltung dargestellt.

Oft wird die Verteilung vor dem Benutzer verborgen. Deshalb ist im Entwurf mitunter eine "transparente" Spezifikation der Verteilung für die Aktionsschicht angebracht. Es ändert sich nur die Sicht der Benutzer auf das System. Die Verteilung wird nach wie vor sowohl konzeptionell als auch im Pflichten- und Lastenheft verankert. Dem Benutzer erscheint das System als monolithisches System. Ihm werden Dienste angeboten.

Der Kollaborationsrahmen kann als Arbeitsblatt dargestellt werden. Ein typisches Arbeitsblatt kann z.B. für Trader-Architekturen in einer e-Learning-Anwendung wie z.B. http://damit.dfki.de wie

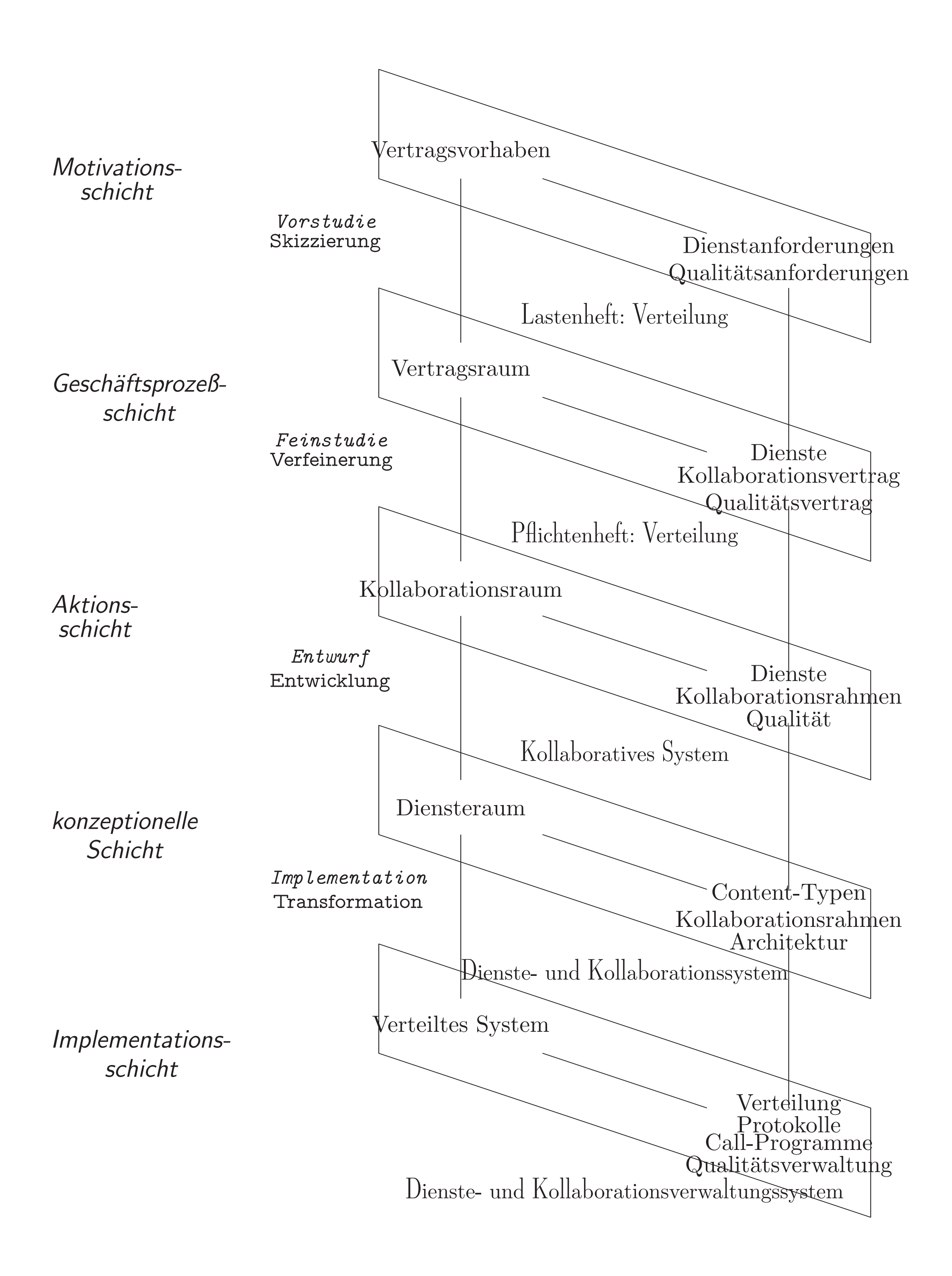


Bild 62: Die Arbeitsprodukte im Abstraktionsschichtenmodell für die Verteilung

folgt zusammengestellt werden:

Trader-Kollaborationsrahmen	Kommunikation	Kooperation	Koordination
Architektur	Asynchroner Client	Kooperationsmanager, Content-Suite-Management-System	Koordinationsmanager
Formation			
Modell	Protokoll: Trader $\bowtie$ P2P	Kontrakt, TA	FIFO Abarbeitung
Release	Login	Hierarchisch	Sharing, Lebensspanne
Datenaustausch	Stub, Fehleraufzeichnung	Öffentlicher, persönlicher Raum	Replica der Suiten
Interplay	Ereignis, offener Raum	reaktiv, proaktiv	Handeln, buchen, bezahlen
Berechnung			
Modell	Message passing	Trader	FIFO
Programme	IP5	Workspace, Nutzerabrechnung	Session-Manager
Datenaustausch	Push through, Token-basiert	Partielles 2-Phasen-Locking	Halb-synchron; volle Synchronisation mit DBS
Interplay	IP5 sequencing	auf Anforderung	2-Phasen-Commit

Verteilte Datenbanksysteme als spezielle kollaborative Systeme

Eine verteilte Datenbank ist eine inhaltlich zusammenhängende Datenbank, die auf mehreren physisch unabhängigen Knoten (Rechner, Speichermedien) verteilt wird. Die auf den Knoten abgelegten Partitionen der Datenbank können dabei auch nicht separiert voneinander sein (Datensharing). Ein verteiltes System ist gekennzeichnet durch

- eine Anwendungsschnittstelle für verschiedene Endbenutzer,
- eine Validierungsfunktion zur Analyse der Datenanforderungen,
- eine Transformationskomponente zur Berechnung der Anforderungen an die Komponenten,
- eine Anfrageoptimierung, die die Verteilung berücksichtigt,
- ein Input/Output-Interface für die Daten,
- eine Formatierungsfunktion zur Anpassung der generierten Daten an die Benutzeranforderungen,
- ein Sicherheitsmanagement, um Datensicherheit zu gewährleisten,
- Backup- und Wiederanlauffunktionen,
- eine Datenbankadministration,
- eine Steuerung für den konkurrierenden Zugriff über das Netz und
- eine Transaktionsverwaltung.

Damit besteht ein verteiltes DBMS aus *Rechnern*, denen *Knoten* zugeordnet sind, einem *Kommunikationsnetzwerk* zur Verbindung der Knoten, aus einem *Netzwerk-Hard- und Software*, aus *Transaktionsprozessoren* (TP) und aus *Datenprozessoren* (DP).

TP
DP
Lokales DBMS

Kommunikations-
netzwerk

TP
DP
Lokales DBMS

Bild 63: Grundsätzliche Architektur verteilter DBMS

Die verteilte Datenbank präsentiert sich gegenüber den Endbenutzern bzw. Anwendungsprogrammen wie eine zentrale Datenbank. Dieses Ziel erfordert das Verstecken aller 'störenden' Aspekte. Die Lösung besteht in der Realisierung eines ('integrierenden' und 'homogenisierenden') globalen Schemas. Deshalb sind die Verteilung der Daten, inklusive der Kopienhaltung (d. h. der *Partitionierung*[16] und *Allokation*), ebenso wie die strukturellen und semantischen Heterogenitäten (mittels *Schematransformation* bzw. *-integration*) zu verstecken. Aus Performanz- und Sicherheitsgründen werden dabei dieselben Daten an verschiedenen Knoten redundant gespeichert (*redundante Allokation*). Informationen des gleichen Typs werden ggf. an verschiedenen Knoten verschieden dargestellt, z. B. anders strukturiert (*strukturelle Heterogenität*) bzw. mit anderen Bedeutungsinhalten (*semantische Heterogenität*). Eine andere Lösung ist die *Partitionierung* globaler Relationen, indem logisch an sich zusammengehörende Daten in homogener Form an verschiedenen Knoten gespeichert werden.

Mit dieser Funktionalität kann ein verteiltes DBMS

- eine Anfrage entgegennehmen,
- diese analysieren, prüfen und zerlegen,
- diese Teile den einzelnen Komponenten zuordnen,
- auf verschiedene I/O-Operationen zurückführen,
- die entsprechenden Daten suchen, lokalisieren, lesen und validieren,
- auf dieser Grundlage die Konsistenz, Sicherheit und Integrität prüfen,
- die Daten entsprechend der ursprünglichen Dekomposition validieren und
- am Ende die gewonnenen Daten entsprechend der Anfrage dem Benutzer zur Verfügung zu stellen.

Diese Aktivitäten sind aber für dem Benutzer *nicht sichtbar*. Wir unterscheiden dabei verschiedene Arten von Sichtbarkeit.

Je nach Verteilung der einzelnen Komponenten unterscheiden wir

Einfachknoten-Berechnung und Einfachknoten-Datenhaltung,

Einfachknoten-Berechnung und Mehrfachknoten-Datenhaltung,

Mehrfachknoten-Berechnung und Einfachknoten-Datenhaltung und

Mehrfachknoten-Berechnung und Mehrfachknoten-Datenhaltung.

Die Mehrfachknoten-Berechnung und Einfachknoten-Datenhaltung entspricht im Wesentlichen der Client/Server-Architektur der Workstation-basierten DBMS.

Wir können auf verschiedene Rechner bei Vorhandensein eines Netzes verschiedene Ressourcen verteilen:

[16] Wir verwenden hier den Begriff 'Partition'. In der Literatur wird neben dem Begriff 'Partition' der Begriff 'Fragment' benutzt. Da wir jedoch auf eine disjunkte Überdeckung des Datenbankinhaltes orientieren, ist das Wort 'Partition' eher

Daten: Daten können auf verschiedenen Rechnern abgelegt und auf Anfrage bzw. Abforderung anderen Rechnern zugänglich gemacht werden.

Prozesse: Prozesse können auf verschiedenen Rechnern ausgeführt und über ein Netz zusammengeführt werden.

Steuerung: Die Bearbeitung kann durch verteilte Steuerung der einzelnen Prozesse und des Datenaustausches erleichtert werden.

Dabei kann die Organisation der Verteilung nach Prozeßcharakteristika und Prozeßwissen unterschieden werden:

Umfang des Sharing: In verteilten Datenbanken kann sowohl kein Sharing an Informationen stattfinden als auch Sharing in verschiedenen Stufen. Je größer der Sharing-Anteil, umso kritischer wird die Pflege und umso besser wird die Zugriffszeit auf Fremddaten.

Verhalten von Zugriffsmustern: Die Zugriffsmuster über das Netz können *statisch* oder auch *dynamisch* sein. Statische Zugriffsmuster, die sich nicht verändern, sind relativ selten. Dynamische Zugriffsmuster bedingen dagegen einen ständigen Anpassungsprozeß.

Umfang des Wissens über den verteilten Zugriff: Die Information über das Zugriffsverhalten kann *vollständig*, ist jedoch meist nur *partiell*. Je weniger Wissen vorhanden ist, umso schlechter kann die verteilte Datenbank an die Anforderungen angepaßt werden.

Grundsätzlich sollen in einer verteilten Datenbank die Benutzer nicht mit der Verteilung direkt konfrontiert sein. Die Verteilung wird deshalb unsichtbar bleiben:

Nichtsichtbarkeit der Verteilung: Die Benutzer wissen nicht, welche Daten auf welche Knoten verteilt wurden.
Wir unterscheiden dabei verschiedene Niveaus von Nichtsichtbarkeit:

- Nichtsichtbarkeit der Partitionierung : Der Benutzer kennt weder die Partitionierung noch die Knoten, sondern kann das System wie eine zentralisierte Datenbank benutzen.
- Nichtsichtbarkeit der Lokalisierung bei sichtbarer Partitionierung : Der Benutzer muß die Partition angeben, nicht aber die Lokalisierung.
- Sichtbarkeit der Lokalisierung und Partitionierung : Der Benutzer muß sowohl die Lokalisierung als auch die Partitionierung angeben.

Nichtsichtbarkeit der Transaktionen: Die Benutzer kennen die Verteilung von Transaktionen nicht.
Durch *remote-Anforderungen* können Daten auch von anderen Knoten, z.T. auch unabhängig und parallel, geholt werden. Es wird durch einige Systeme auch eine *verteilte Steuerung* ermöglicht. Mit einem *Zweiphasen-Commit-Protokoll* wird der Abschluß der Transaktion auch über verschiedene Knoten kontrolliert.

Nichtsichtbarkeit des Ausfalls einzelner Komponenten: Solange ein Ausfall nicht das Funktionieren beeinflußt, erfahren die Benutzer nichts vom Ausfall einzelner Komponenten.

Nichtsichtbarkeit des Funktionierens: Das System hat nach außen das gleiche Verhalten wie ein zentralisiertes System.

Nichtsichtbarkeit der Heterogenität: Das System ist in der Lage, die verschiedenen heterogenen Bestandteile dem Benutzer wie ein einheitliches, auf einem globalen konzeptionellen Schema beruhendes System erscheinen zu lassen.

Daten können auf verschiedene Art wie in Bild 64 partitioniert werden:

Horizontale Partitionierung: Daten werden horizontal zerlegt (d. h. eine Tabelle oder Relation wird tupelweise in verschiedene Teilrelationen zerlegt) und verschiedenen Medien zugeordnet. In Bild 64 wird die Relation R durch Anwendung von Selektionsoperationen in drei Teilrelationen zerlegt, wobei gefordert wird, daß sich die Relation R aus den Teilrelationen durch Vereinigung dieser Teilrelationen wiederherstellen läßt. Damit müssen die Bedingungen α, β und γ als Disjunktion den Wahrheitswert *true* ergeben. Neben Selektionsoperationen können auch andere Operationen der relationalen Algebra verwendet werden. Es wird jedoch im Kontext verteilter DBS exklusiv die Selektion verwendet.

Vertikale Partitionierung: Daten werden vertikal zerlegt (d. h. eine Relation oder Tabelle wird attributweise dekomponiert) und auf verschiedene Medien verteilt. In Bild 64 wurde die Relation R durch Projektion in zwei Teilrelationen zerlegt. Der natürliche Verbund dieser beiden Teilrelationen muß wiederum die ursprüngliche relation R ergeben.

Gemischte Partitionierung: Daten werden sowohl horizontal als auch vertikal zerlegt und auf verschiedene Knoten aufgeteilt. Es werden schrittweise Selektion und Projektion zur Partitionierung angewandt.

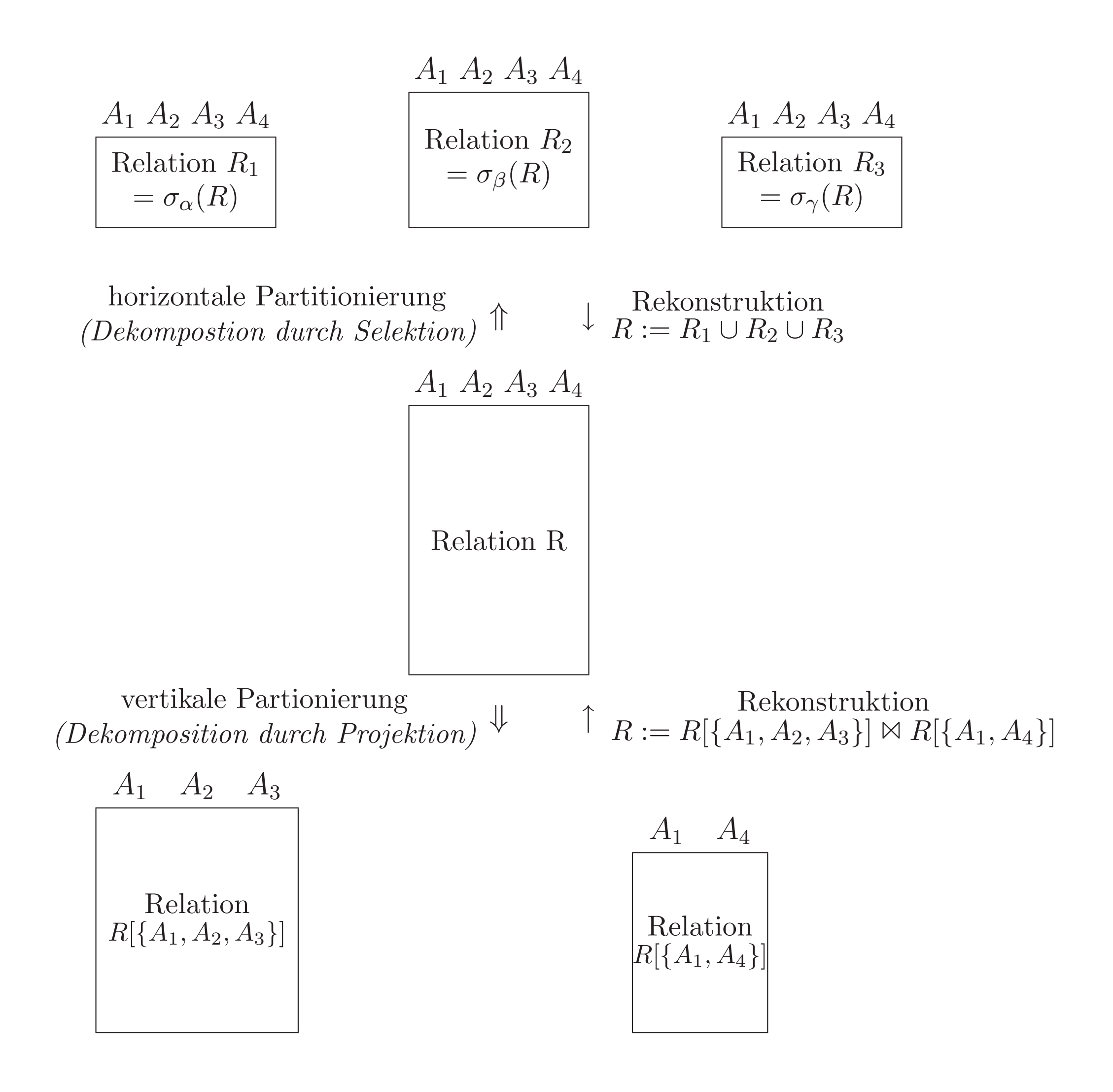


Bild 64: Partitionierungskonzepte

Die Partitionierungstiefe kann bei einer Partitionierung von *keine Partitionierung* bis zu einer *Partitionierung auf Attribut- bzw. Objektniveau* reichen.

Für die Partitionierung sind einige Korrektheitsregeln, in verschiedenen Abstufungen einzuhalten:

Vollständigkeit: In Analogie zur Eigenschaft der verlustlosen Dekomposition bei der Normalisierung können Klassen in mehrere Teilklassen oder anhand von Teilstrukturen in Partitionen zerlegt werden. Eine Eigenschaft eines Objektes kann dabei einmalig oder mehrmalig repräsentiert sein.

Rekonstruierbarkeit: Je nach Zerlegung bzw. Partitionierung existiert eine Funktion ∇ zur Wiederherstellung der ursprünglichen Klassen.

Disjunktheit: Die Partitionen sind entweder disjunkt oder es existiert ein Algorithmus, mit dessen Hilfe gleiche Eigenschaften eines Objektes in verschiedenen Partitionen gepflegt werden können. Meist kann ein solcher Algorithmus über Identifikationsmechanismen definiert werden.

Sobald eine Datenbank partitioniert ist, muß eine Allokation der verschiedenen Partitionen zu den Knoten des Netzes erfolgen. Die Partitionierung und Allokation werden ebenso wie im Falle zentraler DBS in einem Datenbank-Katalog (data dictionary (DD)) verwaltet. Ein zugeordnetes Datum kann dabei repliziert oder einmalig einem Knoten zugeordnet sein. Es können Prozesse für Daten in zwei Extremen unterstützt werden:

Read-only-Zugriff für Replikate: Die Zuverlässigkeit und Effizienz (insbesondere für parallele Zugriffe) ist bei Read-only-Zugriffen auf Replikaten höher. Zugleich entsteht aber ein Update-Problem.

Read-and-write-Rechte für Replikate: Die Zuverlässgkeit und unter gewissen Umständen die Effizienz sinken. Ein Update wird analog zu Triggermechanismen vorgenommen.

Je nach Umfang der Replikation können verschiedene Probleme entstehen. Damit ist für jede Anwendung abzuwägen, inwieweit eine Replikationsstrategie günstig ist.

Art der Replikation:	volle	teilweise	keine
Anfrageberechnung	einfach	$\overset{\text{gleiche Komplexität}}{\longleftrightarrow}$	
DD-Verwaltung	einfach oder nicht existent	$\overset{\text{gleiche Komplexität}}{\longleftrightarrow}$	
Steuerung der Parallelität	mittel	hoch	einfach
Zuverlässigkeit	sehr hoch	hoch	niedrig
Realistische Anwendungen	mögliche Anwendung	realistische Anwendung	mögliche Anwendung

Die Analogie zu Diensteplattformen ergibt hier einen der Implementationsansätze wie in Bild 59 und 60 dargestellt.

In *konventionell realisierten verteilten Datenbanksystemen* wird die Verteilung in den Anwendungen selbst realisiert. Die Anwendungsprogramme können miteinander kommunizieren. Dadurch werden an den Entwurf der Schnittstellen dieser Programme hohe Anforderungen gestellt. In *verteilten Datenbanksystemen* wird die Verteilung über das verteilte Datenbankmanagementsystem übernommen. Die Verteilung der Daten ist für das einzelne Anwendungsprogramm nicht mehr sichtbar.

Allen verteilten Datenbanksystemen ist die Verteilung der Daten auf verschiedene Knoten und die lokale Verarbeitung von Anfragen durch die lokalen Komponenten gemeinsam. Mitunter werden auch verteilte Dateisysteme als verteilte Datenbanksysteme bezeichnet. Obwohl Dateisysteme als Datenbanksysteme der ersten Generation aufgefaßt werden können, haben sie wenig gemeinsam mit Datenbanksystemen. Die Funktionalität von verteilten Datenbanksystemen kann nach der folgenden Tabelle unterschieden werden:

Merkmale verteilter Datenbanksysteme	Homogene eng integr.	Interoperable	Föderierte	Offene Multi-DB
Physische Verteilung der Daten	+	+	+	+
Logische Sicht als eine Datenbank	+	+/-	+/-	-
Nichtsichtbarkeit der Verteilung	+	-	+/-	-
Gemischter DB-Zugang (glob./lok.)	-	-	+	-
Zerlegung glob. Anfragen durch DBMS	+	-	+	-
Lokale Ausführung von Teilanfragen	+	+	+	+
Globales Transaktionskonzept	+	-	+	-

Das verteilte System ist von außen als ein homogenes System sichtbar. Es besitzt ein *integriertes Schema*. Die lokalen Systeme sind *nicht autonom*. Das Transaktionskonzept ist global.

Damit werden Leistungsanforderungen wie im Falle zentraler Datenbanksysteme anwendbar. Daraus resultiert auch die Anwendungsbreite:

- *Hochleistungsdatenbanksysteme* durch Nutzung der Parallelverarbeitung;
- *Fehlertolerante Datenbanksysteme* durch Nutzung der kontrollierten Redundanz;
- *Dezentralisierte Datenbanksysteme* zur Reduktion des Kommunikationsaufkommens und der Abhängigkeit vom Netz.

Mehrrechnerdatenbanksysteme sind eine typische Realisierungsform von homogenen integrierten Datenbanksystemen. Im Wesentlichen sind drei Realisierungsvarianten entwickelt worden:

- In der *Shared-Everything-Architektur* sind sowohl Systempuffer als auch Sperrtabelle global.
- In der *Shared-Disk-Architektur* wird wie in der vorhergehenden Variante die Platten-Peripherie über eine Variante von Bussystemen gemeinsam genutzt. Die einzelnen Anfragen werden lokal durch eigene Rechner mit eigenem Hauptspeicher verarbeitet.
- In der *Shared-Nothing-Architektur* wird ein vollständig verteiltes System aufgebaut, dessen einzelne Systeme durch schnelle Kommunikationverbindungen miteinander verbunden sind.

Föderative Datenbanken folgen dem Besitzer/Benutzer-Prinzip, wobei zusätzlich noch einem Benutzer Leserechte durch den Besitzer verweigert werden können. Sie wirken aufgrund einer Spezifikation der *Kooperation* zusammen. Bei Kopplungen muß auch die lokale Effizienz gewahrt bleiben. Wir unterscheiden dabei

- *singuläre Föderationen*, bei denen die lokalen DBMS heterogen sein können, die jedoch auf einem globalen Schema basieren und dieses für die Berechnungen benutzen, und
- *multiple Föderationen*, bei denen die einzelnen Systeme auch eigene, anderen nicht zugänglich gemachte Daten besitzen, die nicht mehr auf einem globalen Schema beruhen und die über *Exportschemata* miteinander zusammenarbeiten.

Eine Weiterentwicklung von multiplen Föderationen sind *sprachlich gekoppelte Multi-DBMS*. Dazu wird jedoch erst geforscht, so daß hier für den Entwurf nur föderative DBMS betrachtet werden.

Der Entwurf einer föderativen Datenbank kann dabei von folgender Referenzarchitektur ausgehen:

Lokale Schemata sind die Schemata der einzelnen Netzknoten.

Komponentenschemata sind die lokalen Schemata in einer für die Koordinierung aufbereiteten Form. Das Datenbankmodell kann verschieden vom Datenbankmodell des lokalen Schemas sein.

Exportschemata beschreiben die netzweit zugänglichen Daten, die den Teilnehmern einer Föderation zugänglich gemacht werden müssen.

Föderative Schemata fassen die Exportschemata analog zur Sichtenintegration wie oben beschrieben zu einem allgemeinen Schema zusammen. Weiterhin werden Ansätze zur Auflösung von Modellierungskonflikten, statische Daten zur Optimierung, zur Verteilung (Partitionierung, Replikation etc.) erfaßt.

Transformationsprozessoren erlauben eine Abbildung der lokalen Schemata auf die Komponentenschemata.

Filterprozessoren filtern aus den Komponentenschemata die Daten für die Exportschemata heraus.

Konstruktionsprozessoren dienen zur Einbindung der Exportschemata in die oder das föderative Schema.

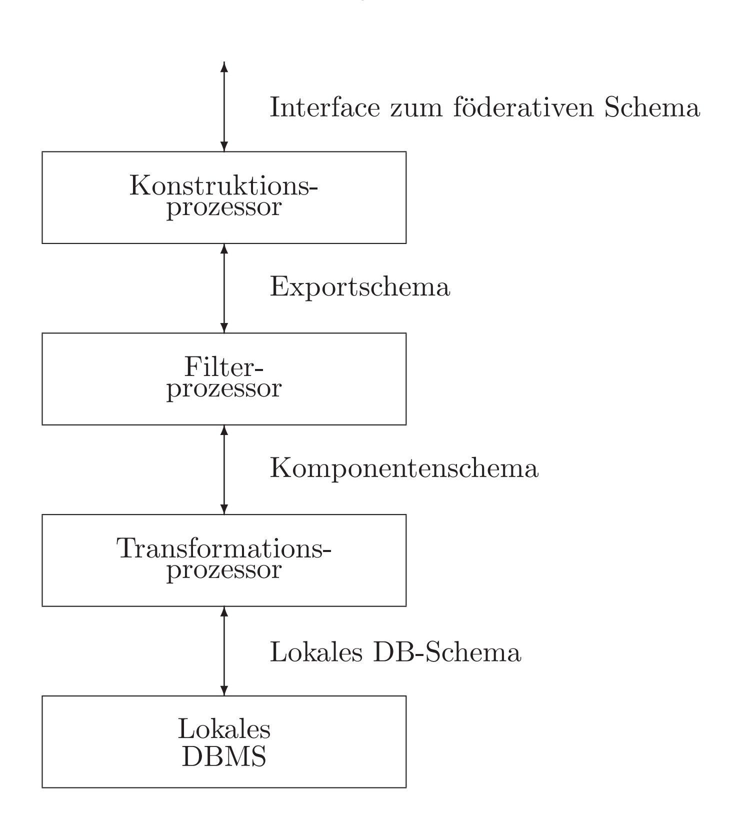


Bild 65: Die Architektur von föderativen Datenbanksystemen

Ein Katalog führt die Metadaten für die DB-Verarbeitung. Im Katalog werden die Namen und Adressen externer Knoten und der DBS, Angaben zur Datenverteilung und Angaben zu Relationen, Sichten, Attribute, Integritätsbedingungen, Benutzern, Zugriffsrechten, Indexstrukturen, Statistiken etc . geführt. Jeder Knoten führt für die lokalen Objekte die Katalogdaten.

Alternativen für die Realisierung eines globalen Kataloges (Verteilungsinformationen, Angaben zu nicht-lokalen Objekten und Benutzern) sind:

- zentralisierter Katalog,
- vollständig replizierter Katalog,
- Mehrfachkataloge: Kombination aus den beiden ersten Ansätzen und
- partitionierter Katalog

Eine weitere Variante ist die Verwendung eines partitionierten Kataloges und die Pufferung (Caching) von entfernten Katalogdaten.

Die beiden wesentlichen Alternativen zur Behandlung veralteter Katalogangaben sind

- entweder eine Verlinkung der Daten, so daß sich der Besitzerknoten vermerkt, an welcher Stelle Katalogdaten gepuffert sind, und invalidiert diese bei einer Änderung, oder

- die Verwendung von Zeitstempeln, so daß bei der Ausführung einer Operation festgestellt wird, ob veraltete Katalogdaten verwendet wurden, und ggf. eine Neuübersetzung und -ausführung mit aktualisierten Daten (wie z. B. im System R*) vorgenommen wird.

Anforderungen an die Namensvergabe sind damit

- eindeutige Bezeichner für globale Objekte: Relationen, Sichten, Indexe usw.,
- Stabilität gegenüber Datenumverteilungen (Migration),
- Unterstützung von Verteilungstransparenz und
- lokale Namensvergabe.

Die Struktur des Namensraums kann entweder global unter Einsatz von Namensservern oder Namenskonventionen konzipiert sein, woduch allerdings ein weiteres Zuverlässigkeitsproblem entsteht, oder hierarchisch sein, wodurch die Knotenautonomie gewährleistet wird, die Netzwerk-Partitionierung toleriert wird und eine Anpassung an das Wachstum einfach ist.

Zur Namensvergabe kann eine dreiteilige Objektbezeichnung

```
[ [<node-id>.]<user-id>.]<object-id>
```

gewählt werden. Damit wird eine lokale Namenswahl durch Benutzer wie in zentralisierten Systemen unterstützt. Verschiedene Benutzer können die gleichen Objektnamen verwenden. Die Referenzierung lokaler Objekte erfolgt wie im zentralen Fall. Diese Lösung erfordert jedoch die Verwendung von <node-id> für externe Objekte. Damit wird die Ortstransparenz verletzt. Eine Änderung der Datenallokation erfordert auch Programmänderungen.

Es existieren eine Reihe Abhilfemöglichkeiten. Durch die Verwaltung von Synonymen (Aliases) für jeden Benutzer, wobei die automatische Abbildung auf vollen Objektnamen durch das DBS erfolgt, kann die Allokation mitgeführt werden. Damit wird bei Datenmigration nur eine Anpassung der Synonymtabellen notwendig oder im ursprünglichem Knoten wird ein Vorwärtsverweis auf die neue Datenlokation gespeichert.

Ein Beispiel eine solchen Lösung ist das System R*. Es verwendet als Syntax

```
<NAME>:: =[<user> [@<user_ node>].] <object_name> [@birth_node]       .
```

Darauf werden Expansionsregeln für systemweite Namen aufgesetzt:

- Ein fehlender <user> wird durch die aktuelle USERID ersetzt.
- Ein fehlender <user_node> wird durch die aktuelle KNOTENID ersetzt.
- Ein fehlender <birth_node> wird durch die aktuelle KNOTENID ersetzt.

Vorteile dieses Zuganges sind Knotenautonomie und die Auswirkungsfreiheit auf Namen und damit bestehende Programme bei Migration eines Objektes. Erkauft werden diese Vorteile mit einer umständlicheren Adressierung, falls das Objekt nicht an Benutzerknoten gespeichert wird, weil mindestens ein Knotenname angegeben werden muß. Durch Synonyme kann hier Abhilfe geschaffen werden.

Das verteilte System wird um eine Routine zur Namensauflösung wie in Bild 66 erweitert. Beteiligte Rechner sind Geburtsknoten ('birth site', meist im globalen Namen enthalten), Katalogknoten ('catalog site') und Speicherknoten ('store site'). Es wird die Replikation (mehrere Speicherknoten) damit unterstützt. Die Trennung von Geburts- und Speicherknoten erlaubt eine Stabilität gegenüber Datenumverteilungen. Der Katalogknoten kann mit dem Geburts- oder Speicherknoten übereinstimmen, wodurch die Kommunikation verringert wird.

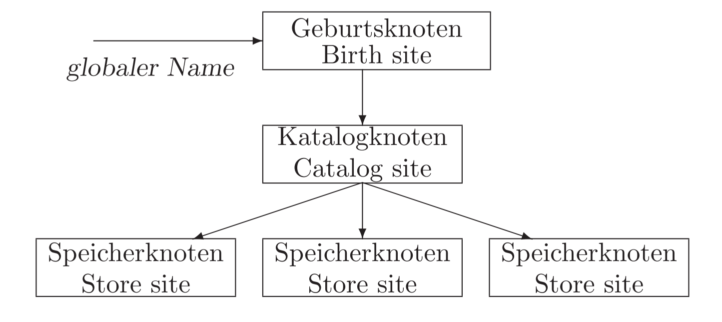


Bild 66: Namensauflösung

In analoger Form findet die Namensauflösung über Synonyme statt. Wie in Bild 67 illustriert, werden Synonyme (Alias-Namen) durch eine Abbildung benutzerspezifischer logischer Namen in vollqualifizierte globale Namen umgewandelt. Die Verwaltung von Synonymtabellen wird durch DBS im lokalen Katalog vorgenommen. Dieser Ansatz findet in vielen kommerziellen Systemen wie z. B. Tandem NonStop SQL, DB2, Oracle, etc. Verwendung.

Der nächste Schritt sind interoperable föderative Informationssysteme wie in Bild 58. Diese Entwicklungslinie läßt sich für interoperable, föderative Systeme fortsetzen.

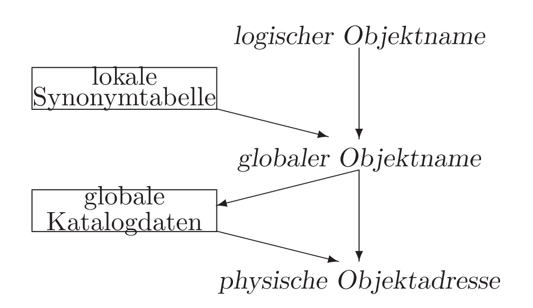


Bild 67: Namensauflösung über Synonyme

Verteilung \ DBMS	Zentrale	Verteilte	Interoperable	Föderative
Datenbankmodell	A	A	B	B
Plattform	A	A	A	B
Replikation/Partitionierung	A	B	B	B

Die verwendeten Zugänge kann man klassifizieren wie in Bild 68 dargestellt.

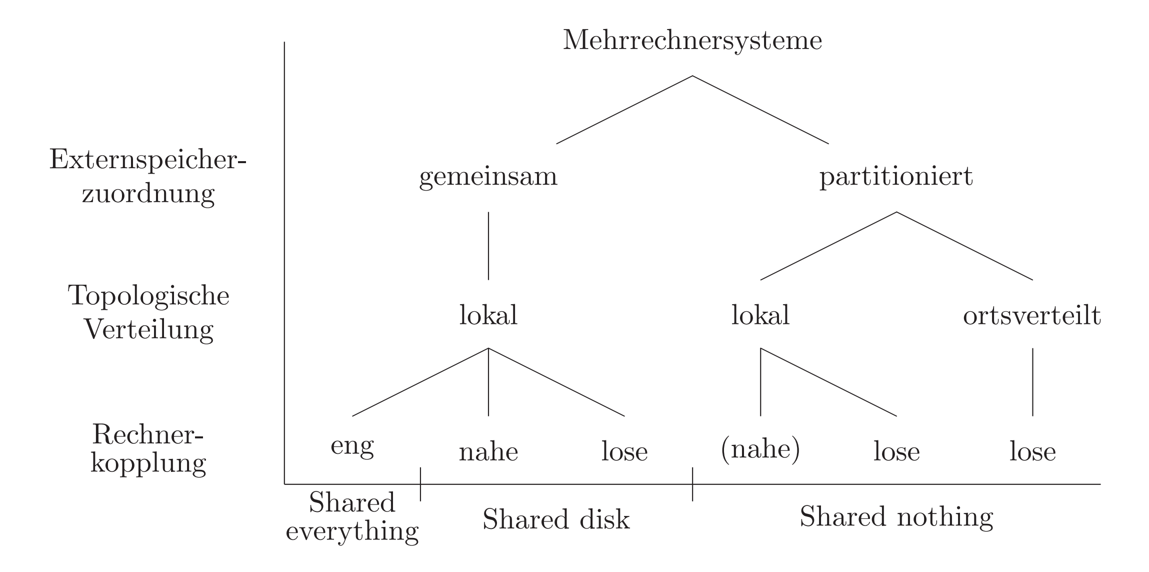


Bild 68: Zugänge für Mehrrechnersysteme

Parallele Datenbanksysteme können in analoger Art und Weise unterschieden werden in:

Shared-everything-Architektur: Mit einem Hochgeschwindigkeitsnetzwerk sind sowohl die Prozessoren als auch die Speicher und die Datenbanken miteinander verbunden. Damit kann eine hohe Universalität durch symmetrisches Multiprocessing erreicht werden. Zugleich sind diese Systeme sehr komplex, schlecht erweiterbar und wenig robust.

Shared-disk-Architektur: Durch ein Hochgeschwindigkeitsnetzwerk werden die Datenbanken und die Einzelrechner miteinander verbunden. Die Einzelrechner benutzen gemeinsam die Datenbanken, sind aber in ihrer Steuerung und Berechnung isoliert.

Shared-nothing-Architektur: Die Rechner verfügen über ihre lokalen Datenbanken, Prozessoren etc. Sie sind über ein Hochgeschwindigkeitsnetz miteinander verbunden.

Die beiden letzten Architekturen haben eine Reihe von Vor- und Nachteilen:

Kriterium	Shared-nothing	Shared-disk
Leistungsfähigkeit	- statische Datenpartitionierung bestimmt Ausführungsort von DB-Operationen - geringe Möglichkeiten zur Lastbalancierung oder Einsparung von Kommunikationsvorgängen - besonders problematisch: ‘dominierende’ Transaktionstypen und DB-Bereiche	- lokale Erreichbarkeit aller Daten wodurch größere Möglichkeiten zur Lastbalancierung entstehen - Kommunikation für Synchronisation und Kohärenzkontrolle - nahe Kopplung kann zur Leistungssteigerung eingesetzt werden; trotzdem höhere Flexibilität zur Parallelisierung
Erweiterbarkeit	- neuer Rechner erfordert physische Neuaufteilung der Datenbank (N $\rightarrow$ N+1) - besonders problematisch für nicht-relationale DBS	- keine physische (Neu-)Aufteilung der DB - direkte Plattenanbindung kann Rechneranzahl begrenzen (‘nachrichtenbasierte’ I/O-Schnittstelle)
Verfügbarkeit	- Partition eines ausgefallenen Rechners zunächst nicht mehr erreichbar - Übernahme/Recovery der betroffenen Partition durch anderen Rechner vorzusehen (ggf. Überlastungsgefahr) - ortsverteilte Replikation ermöglicht schnelle Katastrophen-Recovery	- gesamte DB bleibt nach Rechnerausfall erreichbar - komplexe Crash-Recovery - Erstellung einer globalen Log-Datei
Technische Probleme	- Bestimmung der physischen DB-Partitionierung - verteilte Anfrageverarbeitung - parallele Anfrageverarbeitung - Behandlung replizierter Datenbanken - verteiltes Commit-Protokoll - globale Deadlock-Behandlung - Lastverteilung, -balancierung - Administration - besondere Probleme in ortsverteilten Systemen (Netzwerkpartitionierungen, Knotenautonomie, ...)	- Synchronisation - globale Deadlock-Behandlung - Kohärenzkontrolle - Logging - Recovery - Lastverteilung, -balancierung - parallele Anfrageverarbeitung - Administration

Oft wird eine vollständige Integration von verteilten Systemen angestrebt. Da das Integrationsproblem algorithmisch unentscheidbar ist, kann kein Integrationsalgorithmus existieren. Integrierte Systeme haben ein gemeinsames konzeptionelles DB-Schema. Der DB-Zugriff erfolgt wie im zentralen Fall, womit auch Verteilungstransparenz gewährleistet ist. Damit besitzen die beteiligten DBMS eine eingeschränkte Autonomie. Die einfachste Verwirklichung geht von identischen DBS-Instanzen aus, wodurch ein homogenes verteiltes System entsteht. Beispiele solcher Systeme sind verteilte DBS und Shared-disk-DBS.

Andererseits ist eine vollständige Integration auch nicht das Ziel. Meist ist eine Föderation oder eine Kooperation von Systemen ausreichend. Damit können auch weitgehend unabhängige DBMS mit privaten konzeptionellen DB-Schemata verwaltet werden. Es wird eine partielle Exportierung von Schemainformationen für externe Zugriffe modelliert. Eine Heterogenität ist sowohl bei Datenmodellen als auch bei der Transaktionsverwaltung möglich. Damit entstehen allerdings Probleme mit der semantischen Heterogenität. Eine Verteilungstransparenz ist i. Allg. nur bedingt erreichbar.

Die *Prozessorfunktionalität* gestattet eine weitere Unterscheidung verteilter und Mehrrechner-DBS:

Funktionale Gleichstellung: Jeder Knoten besitzt die gleiche Funktionalität bzgl. DB-Verarbeitung. I. Allg. werden vollständige DBMS in jedem Knoten verwendet. Die Funktionen werden repliziert.

Funktionale Spezialisierung: Die Funktionen werden partitioniert, separiert oder auch spezialisiert. Typische Beispiele sind DB-Maschinen mit Spezialprozessoren für bestimmte DB-Funktionen z. B. für den Verbund, das Sortieren oder auch Kommunikationsfunktionen.

Ein spezielles Beipiel sind Workstation/Server-DBS. Sie werden besonders bei Non-Standard-Anwendungen verwendet. Damit kann eine DB-gestützte Verarbeitung großer, komplex-strukturierter Datenmengen in der Workstation unterstützt werden, insbesondere bei hoher Rereferenz-Wahrscheinlichkeit bei den Daten und bei langen Transaktionen.

Sowohl die Workstations als auch der Server verarbeiten Daten, besitzen eine Steuerfunktionalität und verarbeitende Funktionen. Durch den Workstation-Objektpuffer können Kommunikationsvorgänge eingespart werden. Anfragen und Methoden werden ggf. lokal ausgeführt. Auf dem Server werden globale Aufgaben ausgeführt: Logging, Synchronisation, Externspeicherverwaltung etc.

Die Spezialisierung erschwert Lastbalancierung, Erweiterbarkeit und Fehlertoleranz. Deshalb werden Mischformen aus horizontaler/vertikaler Verteilung verwendet.

Zusammenfassend können wir die Eigenschaften von Mehrrechnersystemen wie folgt vergleichen:

	Parallele DBS	Verteilte DBS	Föderative DBS	Workst./Server-DBS
Hohe Transaktionsraten	++	○/+	○	○
Intra-TA-Parallelität	++	○/+	-/ ○	○/+
Erweiterbarkeit	+	○/+	○	○
Verfügbarkeit	+	+	-	○
Verteilungstransparenz	++	+	○	++
geographische Verteilung	-	+	+	○
Knotenautonomie	-	○	+	-
DBS-Heterogenität	-	-	+	-/ ○
Administration	○	-	-/−	○

Multi-DBMS und Datenbank-Farmen

Verteilte Datenbanksysteme fußen auf lokalen Datenbanksystemen und folgen einer Integrations- und Kollaborationstrategie. Die Integration verwendet kooperierende Sichten in der einen oder anderen Form. Typische Formen der Integration sind:

Global-As-View-Integration (GAV) : Das globale Schema besteht virtuell. Es ist eine Sicht auf die einzelnen lokalen Schemata. Es wird eine Client-basierte Integration angestrebt und eine auf den Einzelsystemen basierende Entwicklung vorgenommen. Anfragen können damit durch Anfrageexpansion über den lokalen Schemata realisiert werden. Damit sind allerdings Integrations-, Semantik- und Pragmatikkonflikte verbunden, die die Praktikabilität dieses Zuganges erheblich einschränken.

Local-As-View-Integration (GAV) : Die lokalen Schemata sind modifizierbare Sichten des globalen Schemas. Die lokalen Schemata sind vollständig innerhalb des globalen Schemas integriert. Die Integritätspflege ist einfacher als beim GAV-Ansatz. Es wird allerdings eine vollständige Integrierbarkeit vorausgesetzt, die in der Praxis selten gegeben ist. Der Berechnungsaufwand zum Abgleich ist erheblich. Es müssen spezielle Kollaborationsverträge realisiert werden, die auch den unterschiedlichen Semantikauffassungen der lokalen Anwendungen Rechnung tragen müssen. Damit entstehen Systeme, die in der Komplexität größer sind als Systeme der Künstlichen Intelligenz. Anfragebearbeitung setzt Logikkomponenten voraus. Außerdem müssen die lokalen Schemata meist restrukturiert werden, womit eine Reprogrammierung erforderlich wird.

Global-and-Local-As-View-Integration (GLAV) : Es werden sowohl eine globale Datenbank als auch die lokalen Datenbanken gepflegt. Dieser Zugang stellt einen allgemeineren Zugang dar, erbt allerdings auch die Nachteile von GAV und LAV.

Die strukturelle Integration ist als Tripel $\mathfrak{I} = (\mathcal{G}, \mathfrak{S}, \mathcal{M})$ bestehend aus einem globalen Datenbankschema $\mathcal{G}$ (über einer Sprache $\mathcal{A}_{\mathcal{G}}$), einer Kollektion $\mathfrak{S}$ von lokalen Datenbankschemata $\mathcal{S}$ (über einer Sprache $\mathcal{A}_{\mathcal{S}}$) und einer Abbildung zwischen $\mathcal{G}$ und $\mathfrak{S}$ definiert.

Die Architektur von Systemen, die über strukturelle Integration realisiert werden, ist in Bild 69 dargestellt.

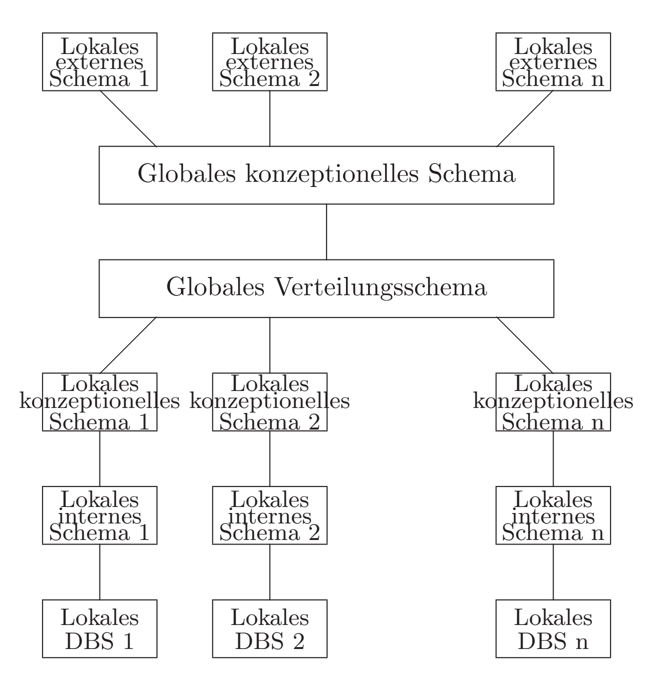


Bild 69: Verallgemeinerung der Dreiebenen-Architektur zu einem verteilten Schema

Offene Mehrdatenbanksysteme sind eine Variante von lose gekoppelten verteilten Systemen mit einem Verteilungsschema, das weder Integrations- noch Abstimmungsforderungen erhebt. Alle Daten müssen jedoch identifizierbar sein über ihre Datenbank.

Im allgemeinen erfordern jedoch verteilte Systeme eine Integration lokaler Schemata. Die Entwicklung des Verteilungsschemas erfordert zusätzliches Wissen über die Anwendungen. Da die Integrierbarkeit unentscheidbar ist, muß das Integrationswissen im Entwicklungsprozeß extra eingebracht und erfaßt werden.

Wir stellen außerdem fest, daß die Integration auf der Grundlage der Sichtenkooperation um ein vielfaches einfacher ist. Sie wird durch Konzentration auf Stern- und Schneeflockenschemata weiter vereinfacht, wenn entsprechende Bücken- oder Scharnierkompositionsoperationen eingesetzt werden.

Die Integration setzt oftmals auf föderierten Architekturen in Analogie zur Architektur in Bild 70 auf. Das Containersystem von Datenbank-Farmsystemen bietet außerdem noch eine explizite Transferkomponente an.

Datenbank-Farmsysteme wurden in [YTS$^+$99] eingeführt. Sie nutzen die Mechanismen von Sichten-Suiten aus:

Informationseinheiten sind verallgemeinerte Sichten. Diese Sichten werden mit einer Funktionalität ausgestattet, die sowohl Retrieval als auch Modifikation von Daten als auch die Arbeit mit den Daten unterstützt.

Container unterstützen den Export und den Import von Daten. Die Container können je nach Bedarf ent- und beladen werden.

Das globale Kollaborations- und Farmsystem stellt entsprechende Dienste auf der Grundlage des Kollaborationsvertrages zur Verfügung.

Datenbank-Farmen erfordern einen Entwurfsschritt mehr. Wir können mit der vorgeschlagenen

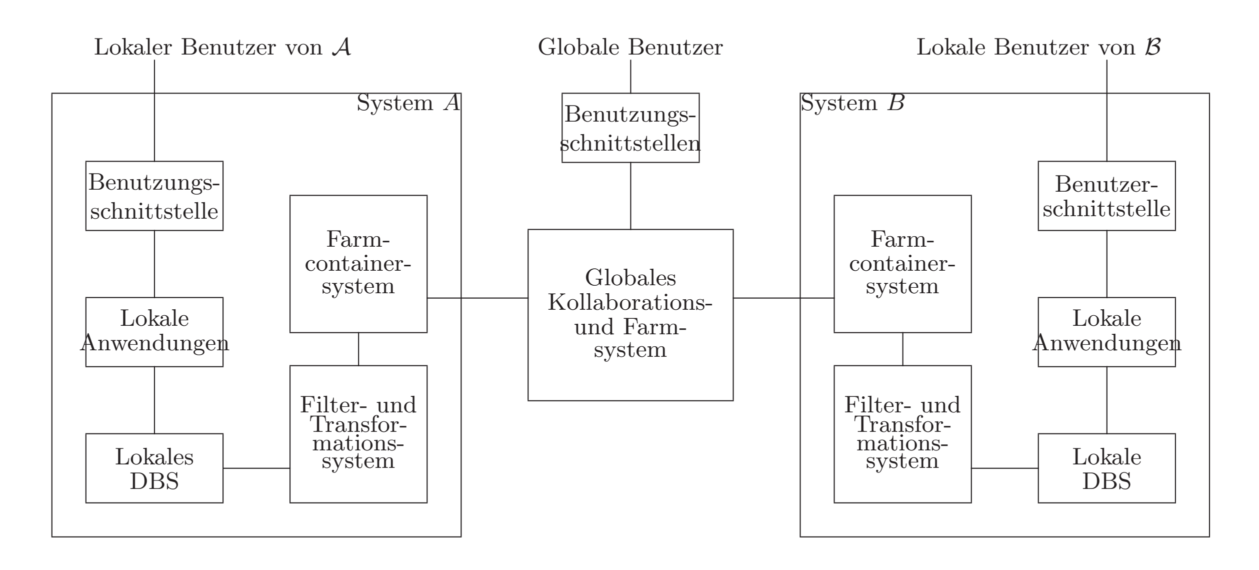


Bild 70: Generelle Architektur von Datenbank-Farmen

z.B. im Denda-Projekt[17] eine Datenbank-Farm realisiert, bei der die Integrität, Sicherheit und der gemeinsame Betrieb auf der Grundlage der Datenbank-Farm-Technologie gesichert wurden.

Datenbank-Farmen müssen auf möglichst einfachen Schemata aufsetzen. Wir nutzen für die einzelnen Teildatenbanken Stern- oder Schneeflockenschemata [LL03, Tha01], die als Komponentensysteme [Rie99, Tha02] verstanden werden können. Die Komponentensysteme werden mit einer Verbindungstechnologie wie oben dargestellt miteinander gekoppelt. Die Kopplung folgt dem Skelett der Anwendung. Die Kopplungsoperatoren dazu sind der Verbund $\bowtie$, der Theta-Verbund Θ, die verallgemeinerte Vereinigung $\sqcup$ und die verallgemeinerte Projektion π.

Aus den Verbindungsoperatoren können die Modifikationsoperationen auf der Grundlage des Datenbank-Farm-Vertrages abgeleitet werden. Der Vertrag versichert auch die Einhaltung der Integritätsbedingungen.

Inkrementelle Farmen von Datenbank-Systemen

Die angeführte Methodik zur Entwicklung von Informationssystemen erlaubt eine inkrementelle Evolution von Datenbanksystemen. Sie ist eine spezielle Form der Systemevolution. Anwendungen im Facility-Management-Bereich erfordern oft die abgestimmte Entwicklung von Farmen von Systemen. Jede einzelne Datenbank hat ihre spezifische Funktionalität und ggf. auch Strukturierung. Zugleich werden Objekte der einen Datenbank an andere Datenbanken übergeben, wobei die übergebenen Objekte nur innerhalb des Kollaborationsvertrages verändert werden dürfen. Meist ist eine Änderung und ein Löschen auf der exportierenden Datenbank untersagt. Mit der "Vererbung" der Objekte an andere Datenbanken werden Objekte mit einer sehr langen Lebenszeit versehen. Die Geschichte der Veränderung wird dann in den vererbten Varianten fortgeschreiben. Facility-Management-Systeme werden in sehr unterschiedlichen Bereichen angewandt, wie im Baubereich, für Patienteninformationssysteme, im Verwaltungsbereich und bei Kunden-Management-Systemen. Sie sind eine spezifische Form der evolutionären Datenbanksysteme. Neben inkrementellen Datenbankssystemen sind auch Archivdatenbanksysteme als evolutionäre Datenbanksysteme bekannt.

[17] Das Denda-Projekt (*Dy*namic *En*vironmental *Da*tabases) war Bestandteil des DFG-Innovationskollegs "Entwicklung und Einsatz dynamischer Datenbanken zur Abschätzung des ökologischen Entwicklungspotentials im Lausitzer Braunkohlerevier". Die unterschiedlichen Gesichtspunkte auf Umweltdaten, die unterschiedliche Granularität der Daten, die unterschiedliche Genauigkeit der Daten und die unterschiedliche zu erreichende Funktionalität erlaubte keine vollständige Integration der Daten. Mit dem Standardisierungkatalog zur integrierten Forführung konnte eine weitgehende kollaborative Arbeitsweise erreicht werden. Das Nichteinhalten der entwickelten Standards im Fortführungsprojekt (Sonderforschungsbereich an der BTU Cottbus) und die Nichtweiterführung des Denda-Projektes war eine entscheidende Ursache für den Zusammenbruch des SFB's. Im Denda-Projekt wurde dagegen erreicht, daß trotz Einhaltung von Autorenrechten auf der Grundlage eines Kollaborationsvertrages eine umfassende Kollaboration durch gemeinsame

Wir betrachten als Anwendung unserer Entwicklungstheorie Facility-Management-Systeme im Baubereich. Die Objekte in diesem Bereich sind interessante Beispiele für *Stern-Typen-Objekte*, die in *Typenschalen* - so wie für Sichten-Suiten dargestellt - schrittweise erweitert werden. Die Entwicklung der Typen in den einzelnen Datenbanksystemen folgt dabei einer Stern-Architektur, in der die Pflege der Daten mit Insert-, Delete- und Update-Formen auf der Grundlage von Trennschichten nach [Tha01] erfolgt.

Für Facility-Management-Systeme ist keine Systematik bekannt. Diese Systeme besitzen einige Phasen:

- In der *Planungsphase* werden die Daten zu Gebäuden, den Bestandteilen und ihren Assoziationen entwickelt. Es entsteht eine Kerndatenbank, die später nicht modifiziert, sondern nur verfeinert wird. Wird eine Änderung in einer späteren Phase erforderlich, dann werden Kopien angelegt und verfeinert.

- Während der *Architekturphase* wird die Gebäudearchitektur entwickelt. Es entstehen zugleich unterschiedliche Sichten auf die gleichen Daten. Daten, die in dieser Phase entstehen, können später ggf. modifiziert werden.

- In der *Bauphase* werden die Architekturentwürfe realisiert. Es werden dabei die Objekte um Baudaten erweitert. Die Daten werden in die Verwaltungsdaten partiell injiziert.

- In der *Verwendungsphase* werden die Daten schrittweise um die Verwendungsgeschichte der Gebäude, ihrer Bestandteile, ihrer Pflege, ihres Ersatzes, ihrer Erweiterung und um Problemaufzeichnungen erweitert.

Eine Architektur für inkrementelle Systeme ist in Bild 71 für das Beispiel von Facility-Management-Systemen skizziert.

Wir nutzen *Zusatzddatenbanksystem* zur Unterstützung des Facility-Management-Systemes. Mit diesem System werden Informationen zu Standards, zur Kundenbeziehung, zur Berechnung, zu Vorschriften usw. in die entsprechenden Systeme injiziert.

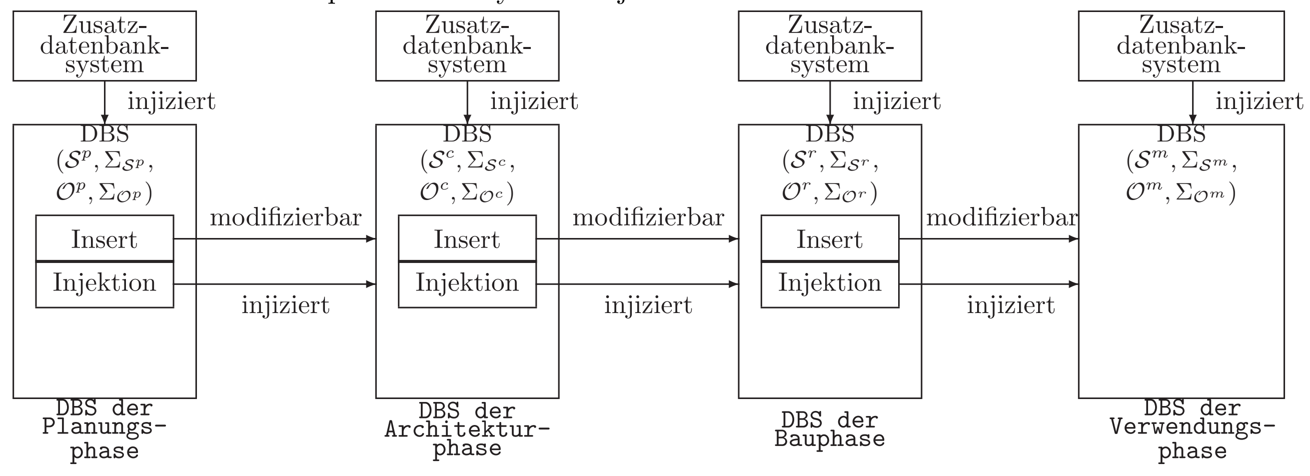


Bild 71: Die allgemeine Architektur für inkrementelle Evolution von Datenbanksystemen

Inkrementelle Systeme verwenden eine spezifische Form der Sichtenkooperation:

Insert-Daten werden den Partnern zur Verfügung gestellt und können verändert werden, wobei die Veränderung ggf. auch im Ursprungssystem durch Überschreibung oder durch das Anlegen einer neuen Version mitgeführt wird. Veränderungen im Ursprungssystem werden i.a. auch mit der Änderungsgeschichte bzw. dem Änderungskuvert geführt.

Injektion-Daten werden den Partnern zur Verfügung gestellt und können nicht verändert werden. Werden diese Daten im Ursprungssystem geändert, dann können die Veränderungen bei den Partnern je nach Vertrag nachgezogen werden oder auch nicht verändert werden. Im letzteren Falle werden die ursprünglichen Daten zur Wahrung der Konsistenz im Ursprungssystem archiviert.

Diese Schemakomponenten werden mit zwei Sichtenkooperationsformen unterstützt:

Injektionsformen unterstützen die Injektion von Daten in Partnersysteme. Die Struktur ist durch eine Sicht $(\mathcal{S}^{inject}, \Sigma_{\mathcal{S}})$ auf die exportierende Datenbank gegeben, die in eine Sicht $(\mathcal{S}', \Sigma_{\mathcal{S}'})$ der importierenden Datenbank eingebettet werden kann. Die Funktionalität $(\mathcal{O}^{inject}, \Sigma_{\mathcal{O}})$ der Sicht des exportierenden Systemes wird ebenso in die Funktionalität $(\mathcal{O}', \Sigma_{\mathcal{O}'})$ des importierenden Systemes eingebettet, wobei alle Modifikationsoperationen allerdings gestrichen werden.

Insertformen unterstützen das Einfügen von Daten des exportierenden Systemes in Daten des importierenden Systemes. Die Struktur $(\mathcal{S}^{insert}, \Sigma_{\mathcal{S}})$ und die Funktionalität $(\mathcal{O}^{insert}, \Sigma_{\mathcal{O}})$ der Sichten des exportierenden Systemes wird dabei in die Struktur $(\mathcal{S}', \Sigma_{\mathcal{S}'})$ und die Functionalität $(\mathcal{O}', \Sigma_{\mathcal{O}'})$ des importierenden Systemes eingebettet. Kann eine Modifikation im importierenden System auf den importierten Daten vorgenommen werden, dann wird die Funktionalität $(\mathcal{O}^{insert}, \Sigma_{\mathcal{O}})$ um entsprechende Versionierungsfunktionen ergänzt.

Verteilte Datenbank-Systeme in der Data-Warehouse-Architektur

Viele Unternehmen haben Unmengen an Daten, ohne daraus ausreichend Informationen und Wissen für kritische Entscheidungsaufgaben ableiten zu können. Die Zusammenführung (Integration) und Verdichtung (Aggregation) von Daten aus mehreren, i. Allg. heterogenen Quellen in einer zentraler Datenbank ist deshalb von großem Nutzen. Damit kann eine schnelle Reaktion auf sich ändernde Marktanforderungen erfolgen. Zugleich entstehen damit komplexe, mehrdimensionale Aggregationsaufgaben, bei denen auch unterschiedliche Zeitpunkte der Datenerhebung, des Einbringens und Validierens zu berücksichtigen sind. Die umfangreichen Anfragen und Datenmengen können meist nur mit parallelen DBS bewältigt werden.

Ein Data-Warehouse (oft auch als *Daten-Warenhaus* bezeichnet) stellt die nächste Entwicklungsetappe für breit benutzbare Datenbanksysteme dar. Im Prinzip kann man sich *Farmen von Datenbanksystemen* vorstellen, die durch eine Art Warenhausverwaltung den Bedürfnissen von Kunden angepaßt und konfektioniert verkauft werden. Dabei wird nicht nur eine Datenbank an sich vermarktet, sondern ein Datenbanksystem mit einer entsprechenden Funktionalität.

Die bereits entwickelte Technologie für benutzerfreundliche Oberflächen kann dabei angewandt werden. Insbesondere sind dabei Methoden von *executive information systems (EIS), on line analytical processing (OLAP), decision support systems (DSS)* anwendbar. In erster Näherung ist dabei ein Datenbank-Warenhaus eine Farm von Datenbanken, die durch *data mining* Werkzeuge einem Benutzer die Auswertung vorhandener Daten ermöglicht. Damit ergibt sich die in Bild 72 dargestellte Architektur.

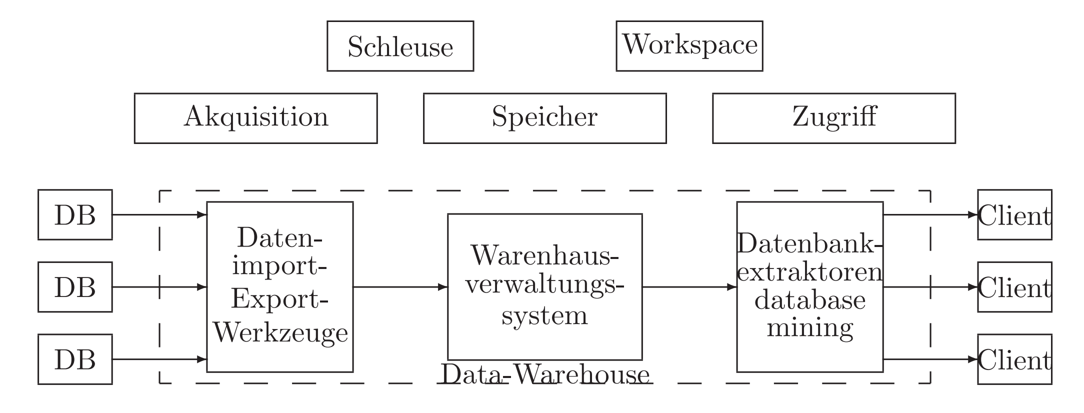


Bild 72: Die drei Komponenten eines Datenbank-Warenhauses

Das Input-Interface kann auch als *Legacy-Interface* bezeichnet werden. Es werden sehr unterschiedliche Datenbanken zum Einsatz kommen, die z. T. auch schon vor längerer Zeit entwickelt werden

und auf anderer technologischer Grundlage basieren (bis hin zu Datenbanksystemen der ersten Generation). Der Speicher dient der Ablage dieser Daten auf einheitlicher technologischer Plattform.

Während der Entwicklung eines Datenbank-Warenhauses sind unterschiedliche *Akquisitionsprobleme*, die bereits für föderative Systeme in Ansätzen auftreten, zu lösen:

Gleiche Information in verschiedenen Datenbanken: Sind Daten in mehreren Systemen vorhanden, dann treten sowohl Konsistenzprobleme, als auch Integrations- und Beschreibungsprobleme auf.

Historische Information: Daten sind meist ohne einen Hinweis auf den letzten update oder auch ihre Gültigkeit abgelegt. Beim Vergleich von Datenbeständen ist aber eine Information über das Alter von Daten sinnvoll.

Korrektheit der Ausgangsdaten: Unterschiedliche Systeme können sehr unterschiedlichen Qualitätsansprüchen genügen.

Unterschiedliche Vollständigkeit der Ausgangsdaten: In den verschiedenen Systemen können zum gleichen Themenkreis Daten in verschiedenem Umfang existieren.

Unterschiedliche Funktionalität der Ausgangssysteme: Information kann sowohl in den Daten der Ausgangssysteme vorhanden sein als auch durch die Funktionalität der Ausgangssysteme extrahierbar sein.

Unterschiedliche Semantikintegration der Ausgangssysteme: Da auch die Ausgangssysteme mit unterschiedlichen Modellierungsmethoden entwickelt wurden, ist auch die Semantik des Anwendungsgebietes auf unterschiedliche Art und mit unterschiedlicher Vollständigkeit abgelegt. Deshalb sind hier auch Probleme zu lösen, die für die Sichtenintegration typisch sind. Ein schwieriges Problem ist z. B. die Behandlung und Interpretation von Nullwerten.

Mit der Entwicklung der *Akquisitionskomponente* sind verschiedene Probleme zu lösen. Eigentlich kann die Akquisition einfach beschrieben werden. Man *wählt* die benötigten Daten *aus*, *lädt* sie und *integriert* sie in das Datenbank-Warenhaus. Damit sind die unterschiedlichen Quellen zu *mischen*, die Daten zu *säubern* und zu *standardisieren*.

Die ***Datenextraktion*** schließt das Streichen oder Gewinnen von Daten von einer Quelle mit ein. Dazu ist auch eine Informationsverarbeitung notwendig. Damit wird auch eine entsprechende Prozessorleistung notwendig.

Die ***Datensäuberung*** basiert auf einer *Qualitätskontrolle* der Daten und schließt die *Identifikation* zu säubernder Daten mit ein. Alle Typen von Datenproblemen, die sonst für Nullwertprobleme üblich sind, treten auf: *inkonsistente, fehlende, unlesbare, falsche, temporär unerreichbare, partiell falsche* Daten sowie einfache Schreibfehler.

Die ***Datenformatierung*** ist notwendig, weil oft weder die Formate, noch die unterlegten Datentypen, noch die Anordnung der einzelnen Elemente den Anforderungen entsprechen. Man vergleiche die Vielfalt von Darstellungen für Kalenderdaten.

Das ***Mischen*** von Daten ist zur Verminderung der *Redundanz* im Warenhaus notwendig. Dabei treten z. T. jedoch erhebliche Integrationsprobleme auf. Die Schreibweise bzw. die Art der Abkürzungen ist für Texte zu vereinheitlichen.

Die ***Schlüsselanpassung*** ist notwendig, um gleichartige Information herauszufinden, um eine Informationsverarbeitung und Verweise auf analoge Informationen zu ermöglichen.

Die ***Prozeßreinigung*** ist ebenso wie die Datenreinigung erforderlich. Dabei hängt viel von der Qualität der Dokumentation ab.

Die ***Allokation der gewonnenen Daten und Prozesse*** im Warenhaus ist so vorzunehmen, daß diese Daten und Prozesse durch die Benutzer auch effizient und mit minimalem Aufwand benutzbar sind.

Die ***Herkunftskennzeichnung*** dient später im Updateprozeß als Information zur Identifikation der zu verändernden Daten.

Diese Prozesse können nicht innerhalb nur eines Schrittes durchgeführt werden, wie einige Firmen heute glauben machen.

Die *Speicherkomponente* ist für sehr große Datenbanken auszulegen. Damit ergeben sich eine Reihe von Eigenschaften:

Extrem große Tabellen: Die Tabellen übertreffen in ihrer Größe und auch Komplexität oft die derzeitigen Möglichkeiten.

Hohe Verflechtung der Daten: Die Daten sind oft hochgradig ineinander verwoben. Oft werden auch Makrodaten (Daten, die aus den Mikrodaten der Datenbanken gewonnen werden) neben Mikrodaten verwaltet.

Ad-hoc-Zugriff überwiegt: In Datenbank-Warenhaus-Anwendungen überwiegt der Ad-Hoc-Zugriff auf die Daten. Zugleich sind jedoch die Benutzer nicht in der Lage, ihre spezifischen Anfragen in einer abstrakten Notation zu formulieren. Deshalb muß eine Variantenvielfalt von Anfragen vorgehalten werden. Es gibt jedoch bereits auf dem Markt einige Umformer von natürlichsprachigen Anfragen.

Gleichzeitiger Zugriff auf viele Tabellen: Obwohl der Verbund von Tabellen bei zentralen Datenbanksystemen adäquat unterstützt wird, ist der Verbund von heterogenen und verteilten Tabellen nach wie vor ein schwieriges Problem.

Benutzer greifen im read-only-Modus zu: Im Warenhaus überwiegt der Lesezugriff. Damit wird die Transaktionsverwaltung wesentlich vereinfacht.

Daten werden aus unterschiedlichen Quellen periodisch modifiziert: Mit einer periodischen Modifikation kann eine Konsistenz von Daten aus unterschiedlichen Quellen nicht ohne weitere Funktionen unterstützt werden. Damit ist allerdings ein entsprechendes Schichtenkonzept neben den Kollaborationskonzepten zu entwickeln.

Daten sind abhängig von ihrer Geschichte: Daten werden in unterschiedlichen Quellen in unterschiedlicher Gültigkeit gehalten. Außerdem ist es oft nicht möglich, Daten unterschiedlicher Quellen mit einem Gültigkeitsabgleich zu versehen.

Großer Zugriffsumfang: Mit Datenbank-Warenhaus-Anwendungen entstehen für bestimmte Daten Zugriffslawinen. Typische Beispiele dafür sind Daten, die aktuellen Anforderungen z. B. in Informationsdiensten genügen müssen wie z. B. dem Samstagsknick bei Fußballdaten. Um größere Zugriffsmengen zu überstehen, empfiehlt sich eine größere Verteilung und eine mehrstufige Sichtenarchitektur.

Für die *Zugriffskomponente* sind eine Reihe von schwierigen Problemen zu lösen.

Der ***Zugriff ist unterschiedlich.*** Er entspricht den Anforderungen einfacher, zufälliger Benutzer und auch denen von Profis. Damit sind eine Reihe unterschiedlicher Zugriffskomponenten je nach Anwendungsfall vorzusehen.

Einfache ***Ad-Hoc-Schnittstellen*** wie die traditionellen Anfragemanager (QBE, QBF, QMF, MS-Query) und Anfragemanager anderer Produkte (Excel, ...), sowie die Reportgeneratoren von verschiedenen Datenbanktools.

Auf das Warenhaus ***zugeschnittene Zugriffsmethoden*** zur einfachen Arbeit mit dem Warenhaus.

Ausgeklügelte ***Auswertungsprogramme*** zur Visualisierung von Zusammenhängen, zur statistischen Analyse, zum Surfen in Querverweisen etc. mit Methoden der künstlichen Intelligenz und zur Simulation.

Autonome ***externe Anwendungen*** wie z. B. Geschäftsanwendungen.

Durch ***Rückkopplungskomponenten*** kann auch auf die Ursprungssysteme bei auftretenden Problemen durchgegriffen werden.

Einfache ***Ad-Hoc-Schnittstellen*** sind für den schnellen, intuitiven Zugriff auf die Datenbestände ein absolutes Muß.

Ausgeklügelte ***Programmpakete,*** die bereits für statistische Anwendungen und zur Visualisierung existieren, sollten integrierbar sein.

Einfache ***Schnupperangebote*** sollen zur Benutzung des Warenhauses verleiten. Die Akzeptanz des Warenhauses kann dadurch verbessert werden.

Schnittstellen zu bereits existierenden Systemen, die einen Datenstrom des Warenhauses verarbeiten sollen, sind zu entwickeln.

Die ***Aktualität*** der Daten ist auszuweisen. Daten unterliegen ebenso wie Produkte einem Verfallsprozeß. Eventuell will ein Benutzer erst über die Aktualität von Daten eine Information erhalten, ehe er diese Daten anfordert.

Ein variabler ***benutzerabhängiger Arbeitsraum*** erleichtert einem Benutzer zu seinen letzten Recherchen zurückzukehren und auch evt. die stattgefundenen Veränderungen seit seiner letzten Recherche zu erkennen. Durch einen entsprechenden Arbeitsraum und seine Verwaltung entsteht ein zusätzlicher Overhead.

Datentypen besitzen oft eine eigenständige Dimensionierung. Mit dieser Dimensionierung sind Aggregationsfunktionen anders anzuwenden sowie ggf. auch nicht sinnvoll anwendbar. Die unterschiedliche Granularität bei der Aggregation führt auch zu Operationen wie ‘Drill-down’ (feinere Dimensionen), ‘Roll-up’ (gröbere Dimensionen), ‘Slice’ (Selektion) und ‘Dice’ (Projektion bzw. Umordnung). Im Einzelnen kann man folgende Dimensionen unterscheiden:

Räumliche Dimension: Daten können eine geografische oder geometrische Verteilung in entprechenden Räumen besitzen, z. B. Ort - Region - Bundesland - Land.

Zeitliche Dimension: Daten werden zu unterschiedlichen Zeiten erhoben, validiert und in die Datenbank eingebracht. Eine typische Zeitdimension ist Tag - Woche - Jahr, die mit der Dimension Tag - Monat - Quartal - Jahr verknüpft sein kann.

Anwendungskategorie bzw. -dimension: Anwendungsobjekte können unter unterschiedlichen Gesichtspunkten gruppiert werden, z. B. ein Produkt nach Produktgruppen und diese nach Branchen.

Diese Dimensionierung sowie Mißverständnisse bei der Anwendung des Sichtenkonzeptes haben zu eigenständigen Entwicklungen ‘mehrdimensionaler’ Datenbanken geführt. Relationale Tabellen mit N Attributen können auch als N-dimensionale Gebilde verstanden werden. Verknüpfen relationale Tabellen M verschiedene Objekte anderer Tabellen miteinander und ordnen diesen Assoziationen Werte zu, dann kann man diese Tabellen durch Relationship-Typen mit M Komponenten verstehen. Die Tabelle

```
Verkauf (KundenNr, FName, ProduktNr, Datum, Menge, Umsatz)
```

kann zu einer 3-dimensionalen Assoziation

```
Region - Branche - Zeit
```

aggregiert werden, die die Anzahl der Verkäufe nach Regionen, Branchen und Zeitpunkten wie in Bild 73 darstellt. Der Relationship-Typ basiert dabei auf den Relationen

```
KUNDE (KundenNr, KundenName, Geschlecht, Alter)
PRODUKT (ProduktNr, PName, PGruppe, Branche, Hersteller, Farbe, Preis)
ZEIT (Datum, Tag, Monat, Quartal, Jahr)
FILIALE (FName, Ort, Land, Region)
```

und besitzt außerdem die Attribute

`Anzahl, Umsatz` .

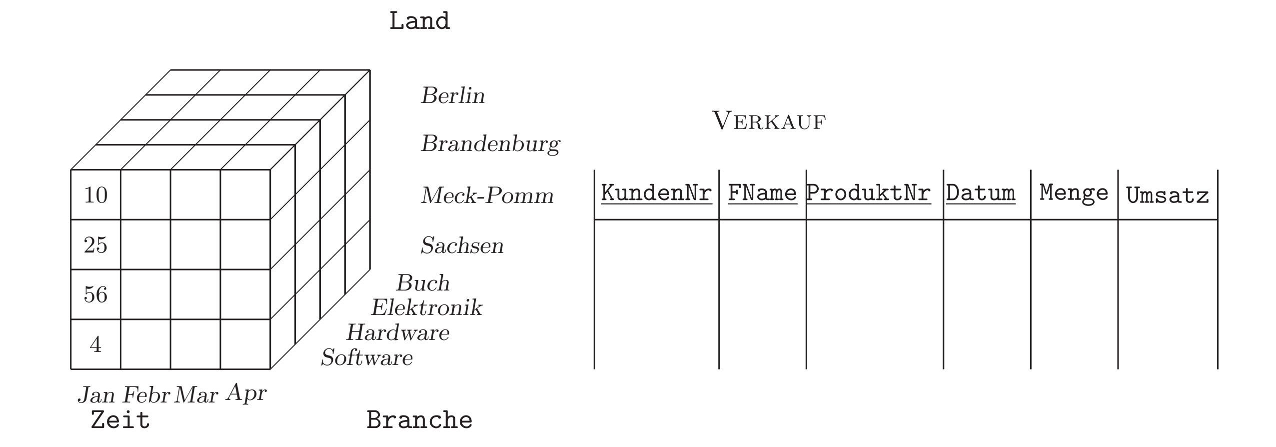


Bild 73: Die 3-dimensionale Aggregation von VERKAUF

In der Aggregation für 'Januar', in der Branche 'Software' und im Land 'Sachsen' wurden z. B. 4 Produkte bestellt.

Aufgrund der Tabellendefinitionen kann leicht über unterschiedliche Aggregationen der Tabelleninhalt von VERKAUF zusammengefaßt werden. Damit können auch weitere Aggregationen über den bereits betrachteten SQL-Anfragen definiert werden.

Beispielsweise wird die Anfrage *"Welche Software-Hersteller wurden von weiblichen Kunden in Berlin im 1. Quartal 2003 favorisiert?"* durch die folgende Anfrage berechnet:

```
select p.Hersteller, sum (v.Anzahl)
from Verkauf v, Filiale f, Produkt p, Zeit z, Kunde k
where z.Jahr = 2003 and z.Quartal = 1 and k.Geschlecht = 'W' and
         p.PGruppe = 'Software' and f.Land = 'Berlin' and
         v.Datum = z.Datum and v.ProduktNr = p.ProduktNr and
         v.FName = f.FName and v.KundenNr = k.KundenNr
group by p.Hersteller;
```

Analog kann Roll-up durch Elimination eines Attributes aus der Group-By-Klausel berechnet werden. Drill-down wird durch Hinzufügen eines Attributes in der Group-By-Klausel berechnet.

Durch eine Definition von Sichten und deren Materialisierung kann ein Datenbank-Warenhaus zusammengestellt werden. Demzufolge kann man zwischen *Mikro-Daten* der Datenbank und den *Makrodaten* des Warenhauses unterscheiden.

9 Die Dokumentation der Spezifikation

Nicht Kunst und Wissenschaft allein,
Geduld will bei dem Werke sein.
Goethe, Faust, Erster Teil, Hexenküche, Mephistopheles

Entwicklungsdokumente

In unserem Vorgehensmodell werden folgende Dokumente schrittweise erarbeitet, verfeinert bzw. dienen als Basis für die Entwicklung anderer Dokumente:

Lastenheft: **(L)** Zur Dokumentation verwenden wir ein erweitertes Lastenheft.

- Bestimmung der Ziele und des Produkteinsatzes: **(LZ)** Es werden die Ziele der Produktentwicklung beschrieben. Die Aufgaben und Einsatzfelder des Informationssystemes werden kurz beschrieben. Wesentliche Charakteristika des Produkt-Einsatzes, wie z.B. Anwendungsbereiche, Zielgruppen und Betriebsbedingungen (phys. Umgebung, Betriebsumgebung und Betriebszeit), werden ebenso festgelegt wie auch die Produkt-Umgebung mit einer Beschränkung für Software (Betriebssysteme, Laufzeitsysteme, Fenstersysteme, einsetzbare DBMS, Compiler bzw. Interpreter), die einsetzbare Hardware (CPU, Peripherie, Drucker, ...), die angestrebte Orgware (Netze, Infrastruktur, ...) und die Produkt-Schnittstellen.
- Beschreibung der Produktdaten: **(LD)** Es werden die wesentlichen Gruppen von Datentypen sowie deren Einsatz und Gewinnung kurz beschrieben. Produktdaten werden in Konzepten zusammengefaßt und zu einer Konzeptlandkarte vereint.
- Beschreibung der Produktfunktionen: **(LF)** Es wird die Hauptfunktionalität (Kernfunktionen und Schnittstellen) des Produktes mit den entsprechenden Anwendungsbereichen, Zielgruppen und Adressaten beschrieben.
- Beschreibung des Diskurs: **(LS)** Es werden der Interaktionsraum kurz skizziert und die Benutzer allgemein beschrieben. Außerdem werden sie im Zusammenhang mit den zu lösenden Aufgaben charakterisiert. Wir erfassen damit das Anwendungsgebiet und die Anwendungsschritte.
- Beschreibung der Sichten und der Verteilung: **(LV)** Es wird eine Grobbeschreibung von Sichten auf das Informationssystem mit der Verwendung durch den Benutzer und der Integration mit den Objekten des Datenbanksystems angegeben. Die Sichten beschreiben die Daten in der Form, in der die Benutzer mit den Daten arbeiten, d.h. als Produktdaten und Produktdatenskizze.
- Beschreibung der Produktleistungen und der Qualitätsanforderungen: **(LQ)** Die Leistungsanforderungen an Kerndatentypen und die Hauptfunktionalität und allgemeine Anforderungen an die Produktqualität wie Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit, Übertragbarkeit, Sicherheit, Portierbarkeit und Erweiterbarkeit werden kurz angegeben.
- Aufwands- und Kostenabschätzung: **(LK)** Anhand der Strukturierung, Funktionalität, der Produktdaten und der Diskurse wird eine Grobabschätzung des Entwicklungs-, Installations- und Pflegeaufwandes vorgenommen.
- Weitere Bestandteile **(LW)** beschreiben die Berücksichtigung von Rechten (Privatrecht, Datenschutzrecht (Persönlichkeitsrecht, Recht auf informationelle Selbstbestimmung, Grundrecht auf Datenschutz), die Darstellung des angestrebten Softwareschutzes (Vertragsrecht, Urheberrecht, auch bei Reengineering), die Mitbestimmungsrechte (Arbeitsverfassungsgesetz, Personalvertretungsgesetz, Betriebsverfassungsgesetz), den Verweis auf das Produkthaftungsgesetz, insbesondere im Zusammenhang mit Qualitätsmangement, die Instruktionspflicht für das Fehler-Management, das Vertragsrecht (Mängelhaftung, Erklärungen zum "Stand der Technik") und die vertragliche Regelung der Software-Hinterlegung, z.B. mit einer Hinterlegungsstelle.

Das Lastenheft sollte so detailliert sein, daß eine Vermarktungsabteilung aus dem Lastenheft eine Vermarktungsstrategie erarbeiten kann.

Architektur der Anwendung: **(A)** In unserem Komponentenansatz erhalten Verteilungs- und Föderierungsaspekte eine relativ einfache Lösung. Es wird die allgemeine Architektur der Anwendung als Komponenten-Schema angegeben. Es werden damit die Zusammenhänge der Komponenten dargestellt, die Abgrenzung der Komponenten voneinander, die Kooperationsbeziehungen der Komponenten unteinander und die Art der Vererbung und Steuerung von Strukturierung und Funktionalität einer Komponente durch andere Komponenten. Da wir eine Separation von Gesichtspunkten und allgemeinen Anwendungsfällen anstreben, ist das Komponenten-Schema die Grundlage für die Entwicklung des Informationssystemes in unserem Vorgehensmodell.

Pflichtenheft: **(P)** Das Pflichtenheft für die Entwicklung von Informationssystemen stellt eine Erweiterung des Pflichtenheftes nach dem IEEE SRS (Software Requirements Specification) Standard dar.

Das Lastenheft dient zur Kurzbeschreibung des Pflichtenheftes. Alle Elemente des Lastenheftes, wie z.B. die Beschreibung zum Produkteinsatz und zur Produktumgebung, die Qualitätsmerkmale, die externen Schnittstellen, behalten ihre Gültigkeit.

Zielbestimmung: **(PZ)** Es werden die allgemeinen Entwicklungsziele, die Produktziele, wichtigste Vergleichsprodukte und die Anforderungen an das Produkt kurz und präzise dargestellt. Weiterhin werden Annahmen und Abhängigkeiten innerhalb der Komponenten und der Komponenten untereinander dargestellt. Außerdem sollten Einschränkungen, wie z.B. Entwicklungsrestriktionen und funktionale Anforderungen, ihren Niederschlag finden.

Die Zielkriterien können in Mußkriterien für unabdingbare Eigenschaften, Wunschkriterien für anstrebbare Eigenschaften und Abgrenzungskriterien für nicht anzustrebende Eigenschaften enthalten. Es können externe Schnittstellen erforderlich sein, so daß deren Anforderungen beschrieben werden.

Beschreibung der Strukturierung: **(PD)** Es werden die Struktur und wichtigsten statischen Integritätsbedingungen kurz and allgemein dargestellt. Dazu kann sowohl ein hierarchisches ER-Modell herangezogen werden als auch eine allgemeine Beschreibung mit Mindmap-Techniken. Die Strukturierung des Informationssystems wird in einer Skizze, die auch grobe Typen umfaßt dargestellt.

Beschreibung der Funktionalität: **(PF)** Es wird die Funktionalität durch Arbeitsschritte unterlegt. Darauf aufbauend werden die Geschäftsprozesse beschrieben, die funktionale Anforderungen, die Leistungsanforderungen, die Entwurfsrestriktionen und die Produkt-Leistungen sowohl zeitbezogen als auch umfangsbezogen. Die Beschreibung der Funktionalität beruht auf Geschäftprozessen und den Arbeitsschritten.

Beschreibung des Handlungsrahmens: **(PS)** Es werden die Hauptszenario der Anwendung zu einer allgemeinen Beschreibung des Story-Raumes verdichtet. Jedes Szenario läßt sich dadurch als ein Durchlauf durch den Story-Raum darstellen. Die Grobdarstellung des Story-Raumes durch einen Handlungsrahmen umfaßt die Beschreibung von Szenen, Dialogschritten und die Assoziierung jeder Szene mit einer Sicht und der erforderlichen Funktionalität. Die Beschreibung der Funktionalität erfolgt aus der Benutzungssicht.

Benutzer werden zu Gruppen von Benutzern mit einem Benutzertyp zusammengefaßt und allgemein mit ihren Aufgaben, Rechten, Rollen und Verantwortlichkeiten beschrieben. Die Beschreibung des Handlungsrahmens führt zu einer Darstellung der Stories und der Ereignisse.

Beschreibung der Sichten und der Verteilung: **(PV)** Die Sichtenskizze beschreibt die einzelnen Sichtweisen der Anwender auf das Produkt in allgemeiner Form. Die Elemente der Sichtenskizze sind allgemeine Konzepte aus der Anwendung, die wir als ontologische Einheiten bezeichnen. Die Verteilung wird durch den Vertragsraum grob skizziert und mit einer Darstellung der Dienste, des Kollaborationsvertrages und des Qualitätsvertrages unterlegt.

Ergänzende Festlegungen: **(PW)** Es können Festlegungen zum Präsentationsstil und der Benutzungsoberfläche (Bildschirmlayout, Drucklayout, Tastaturbelegung, Dialogstruktur, Übergabeformate) und zur Qualitätszielbestimmung (Kriterien-Güte-Katalog) getroffen werden. Vorteilhaft ist die Festlegung von globalen Testszenario und von Testfällen mit einer Darstellung des Verhaltens für jeden einzelnen Testfall. Außerdem kann die Entwicklungsumgebung (Software, Hardware, Orgware und Entwicklungsschnittstellen) festgeschrieben werden. Ergänzungen betreffen auch Installationsbedingungen, Schulungen, Wartungsprobleme, Normen, Vorschriften, Patente, Lizenzen und das Benutzerhandbuch (mit Anwendungsszenarien und Anwendungsbeispielen).

Als Anhang zum Pflichtenheft sind Definitionen der Fachbegriffe auf der Grundlage eines Glossar bzw. Begriffslexikons üblich. Die Fachabteilungen des Anwenders sind hierbei meist involviert.

Dokumente der Aktionsschicht **(A)** sind

das Anwendungsschema als Skelett **(AD)** mit einer Spezifikation der Anwendungstypen,

die Nutzer-Maschine **(AF)** mit einer Darstellung der Handlungen und der entsprechenden Aktionen,

die Aktionssichten-Suite **(AC)** mit den Sichtenskeletten und den entsprechenden Kerntypen,

das Storyboard mit Szenario, Aufgaben und Benutzergruppen **(AS)** zur Darstellung der Interaktivität, wobei der Repräsentant eines Benutzertyps - Akteur - mit einem Profil und Portfolio beschrieben wird, mit seinen Daten- und Prozeßanforderungen angegeben sowie seinem Polaritätenprofil spezifiziert wird, und

das kollaborative System mit Diensten, Kollaborationsrahmen und Qualitätsanforderungen **(AV)** des Kollaborationsraumes.

Die Dokumentation der konzeptionellen Schicht **(K)** basiert auf dem

dem ER-Schema **(KD)** mit den ER-Typen,

der Workflow-Maschine **(KF)** mit den Workflows bzw. Workflow-Feldern und Programmen und den unterlegten Prozessen,

dem Drehbuch **(KS)** mit dem Szenenraum und den einzelnen Dialogschritten,

die Content-Typen **(KC)** der Sichtenschemata mit entsprechenden Typen und

den Dienste- und Kollaborationsraum **(KV)** des verteilten Systemes, die auf entsprechenden Content-Typen beruhen, eine Darstellung der Architektur und auch die Qualitätskontrollmechanismen einschließt.

Die Dokumentation der Implementationsschicht **(I)** sollte aus den vorhandenen konzeptionellen Schemata generiert werden. Sie umfaßt

die logischen Schemata mit

dem logischem Schema **(ILD)** mit einem objekt-relationalen oder semistrukturierten Schema und entsprechenden logischen Typen,

den Programmen wie stored procedures, Transaktionen, Trigger **(ILF)** der Datenbank-Maschine,

der Inszenierung, Programme des Dialogverwaltungssystemes und Oberflächen **(ILS)** zur Darstellung der Interaktion und des Präsentationsraumes im Rahmen des Dialogverwaltungssystemes,

den Programmen des verteilten Systemes **(ILV)** mit einer logischen Spezifikation der Verteilung, den gewählten Protokollen, den Call-Programmen und der Qualitätsverwaltung,
sowie

der logischen Sichten-Suite **(ILC)** mit den entsprechenden Sichtenschemata zur Unterstützung der Content-Typen

die physischen Schemata **(IP)** der Implementation, sowie

die Sicherheitsarchitektur **(S)** und

das Fehlerverwaltungsschema **(F)**, um eine Robustheit des Systemes zu sichern.

Schrittweise Entwicklung von Anwendungen

Eine Methodik bedarf neben einer Theorie auch eine Systematik, die auch einer strengen Bewertung der Software-Entwicklung standhält. Deshalb wurde die Co-Design-Methodik einer Bewertung durch das SPICE 2.0 Framework im Sommer 2003 unterzogen. SPICE unterscheidet fünf Niveaus der Reife des Software-Entwicklungsprozesses:

Niveau 1: (Definierter Prozeß) Der Entwicklungsprozeß ist mit einer Schrittfolge definiert. Alle erforderlichen Produkte sind definiert und entstehen im Entwicklungsprozeß.

Niveau 2: (Beherrschung des Entwicklungsprozesses) Der Entwicklungsprozeß selbst ist beherrschbar, überschaubar, verwaltbar und geplant. Es sind alle Ziele erfaßt. Es werden die Ziele und deren Erfüllung in Einzelschritte planbar und kontrollierbar. Risiken und pragmatische Annahmen werden mit erfaßt. Die Verantwortlichkeit ist klar geregelt. Die Parteien des Entwicklungsprozesses sind bekannt. Deren Rollen und Rechte werden mit verwaltet. Die Ressourcen und die Schnittstellen werden mit verwaltet.

Niveau 3: (Standardisierung des Entwicklungsprozesses) Es sind Standards, Referenzmodelle, Strategien der Entwicklung, Festlegungen für das Fällen von Entwicklungsurteilen wohl definiert und auch durch entsprechende Anwendungen unterlegt. Der Entwicklungsprozeß ist standardisiert, besitzt eine Audit- und Controlling-Methode, ist als Prozeß etabliert, umfaßt die Aufgabezuordnung für alle Beteiligten, hat klare Anforderungen an die zu nutzende Infrastruktur, verfügt über die entsprechenden theoretischen Grundlagen für die Ausbildung von Beteiligten und kann über eine Verwaltung auch Lösungen integrieren.

Niveau 4: (Vorherschaubarkeit des Entwicklungsprozesses) Der Aufwand für die einzelnen Entwicklungsschritte ist quantitativ erfaßbar und durch Metriken berechenbar. Die Qualität des Produktes und des Produktfortschrittes kann gemessen werden.

Niveau 5: (Optimierbarkeit des Entwicklungsprozesses) Der gesamte Entwicklungsprozeß kann an die unterschiedlichen Kontextbedingungen angepaßt und auch optimiert werden.

Niveau 4 und 5 sind bislang von keiner Entwicklungsmethodik erreicht worden. Es gibt wenige Beispiele für Entwicklungsmethodiken, die bereits das Niveau 1 überschritten haben. Die Co-Design-Entwicklungsmethodik ist eine der wenigen, die bereits das Niveau 3 erreicht hat. Sie basiert auf einer Schrittfolge, in der in jedem Schritt die folgenden Parameter dargestellt werden:

Schritt xy	Teilschritt 1. Teilschritt 2. .... Teilschritt z.
Verwendbare Dokumente	aa1 aa2
Betroffene Dokumente	aa1 aa3 oo (Obligationen)
Erstellte Dokumente Resultate	aa4, aa5 oo (Obligationen) rr1, rr2, rr3
Ziele und Gegenstand	[illegible]

Beteiligte	cc, dd
Theoretische Grundlagen	e, ee, ef
Methoden und Heuristiken	gg
Beginnbedingung	hh
Abschlußbedingung	kk

Dieser allgemeine Rahmen erlaubt eine genaue Erfassung des Entwicklungsfortschrittes. Gleichzeitig entsteht eine Dokumentation der Gesamtentwicklung, die auch die Einzelentscheidungen sichtbar macht. Es werden weiterhin die Heuristiken, die zu bestimmten pragmatischen Annahmen führen, mit erfaßt. Damit kann auch zu einem späteren Zeitpunkt nachvollzogen werden, warum ein Entwicklungsschritt zu einem bestimmten und nicht zu einem anderen Resultat führte.

Es wird neben der Dokumentation der Entwicklung auch eine Erfassung der verbliebenen und z.T. neu entstehenden Arbeitsaufgaben vorgenommen. Dazu gehören insbesondere auch Listen von Obligationen, die beim Entstehen weiterer Obligationen erweitert werden und die beim Erfüllen von Obligationen verkürzt werden.

Weiterführende Literatur

Eine Literaturliste zum Co-Design von Strukturierung, Funktionalität, Verteilung und Interaktivität würde sehr lang sein. Die Entwicklung von Informationssystemen mit einer integrierten Spezifikation von Strukturierung und Funktionalität wurde bei einer ganzen Reihe von Zugängen der Objektorientierung versucht. Leider [Web95] wurde aber die Objekt-Orientierung zu stark mit Konzepten überladen, ohne auf deren Integrierbarkeit zu achten. Außerdem wurde mit der Orientierung auf die Einzelobjektspezifikation den Belangen der Massendatenverarbeitung zuwider gehandelt. Deshalb fristen objekt-orientierte Datenbanksysteme z.Z. ein Nischendasein. Eine Abkehr von der klassischen Objekt-Orientierung wurde durch die Entwicklung objekt-relationaler Systeme vorgenommen. Eine gute Literaturquelle für objekt-relationale Systemspezifikation kennen wir bislang nicht.

Die klassische Datenbankliteratur ist riesig und fast unübersehbar. Jedes Jahr erscheinen selbst im deutschsprachigen Raum Dutzende neue Bücher. Eine Klassifikation oder gar eine Übersicht erscheint fast unmöglich. Um eine weitergehende Lektüre zu erleichtern verwenden wir deshalb die folgende Klassifikation:

Hauptliteratur zum Co-Design ist das Buch [Tha00].

Literatur zur Strukturierung von Informationssystemen: Die Literatur zur Spezifikation der Strukturierung von Informationssystemen ist in [Tha00] ausführlich zusammengetragen und erläutert. Als weiterführende Literaturquellen empfehlen wir [BCN92, Bis95, Dre95, EN00, Emb98, Gil90, Hal95, Haw90, Mac90, RS97, RG94, Ris88, Sim94, Bek92, Teo89, FvH89].

Literatur zur Funktionalität von Informationssystemen: Es gibt relativ wenige Bücher außer [Tha00], die die Funktionalität von Informationssystemen überhaupt betrachten. Als weiterführende Literatur zu den Zugängen dieses Skripts können jedoch die folgenden Quellen genannt werden: [Alt96, BDK01, Bes95, BP98, Elm92, Emb98, GR94, Mok96].

Literatur zur Verteilung von Informationssystemen: Zur Verteilungsspezifikation im Rahmen der oberen Spezifikationsschichten existiert keine Literatur. Ergänzende Literaturquellen sind dazu [BS97, BL93, BHG87, Bra94, Bro00, CP84, Rah94, Dad96].

Literatur zur Interaktivität von Informationssystemen: Dazu existieren kaum Literaturquellen. Die Literatur konzentriert sich auf die Gestaltung von Graphik. Gute Quellen in der letzten Kategorie sind [ABH97, Hau00, MS95, Hor99, Pae00, Ebe94, Fer95, FS95, Glo96].

Literatur zur Theorie des integrierten Designs ist z. B. [AHV95, Ago86, AD93, BS03, KK93, KE01, LL99, Leo92, LM78, Mai83, MR92, PBGG89, Tha91, Tsa89].

Weiterführende und interessante Literaturquellen sind besonders im Bereich des Drehbuchdesigns zu finden. Wir verweisen hier auf die weiterführende Literatur auf unserer Website.

Die Literatur zur Gestaltung graphischer Benutzungsschnittstellen ist unüberschaubar. Wir benötigen davon - wie bereits erläutert - nur einige zentrale Werke wie [MS95]. Die Theorie der Gestaltungsraster und der Gestaltungsrahmen basiert auf originären Arbeiten von T. Moritz [Mor03]. Die Theorie der Kollaborationsrahmen ist ebenso wie diese Theorie noch nicht publiziert und am Lehrstuhl DBIS entstanden.

Literaturverzeichnis

[AAB+98] M. Albrecht, M. Altus, E. Buchholz, H. Cyriaks, A. Düsterhöft, J. Lewerenz, H. Mehlan, M. Steeg, K.-D. Schewe, and B. Thalheim. Radd - rapid application and database development. compiled readings of papers published in the radd project. BTU Cottbus, Computer Science Institute, 1998.

[ABH97] P. M. G. Apers, H. M. Blanken, and M. A. W. Houtsma. *Multimedia Databases in Perspective*. Springer, Heidelberg, 1997.

[AD93] P. Atzeni and V. De Antonellis. *Relational database theory*. Addison-Wesley, Redwood City, 1993.

[Ago86] M. Agosti. *Database design: A classified and annotated bibliography*. Cambridge University Press, 1986.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, Reading, MA, 1995.

[ALSS03] S. Abeck, P.C. Lockemann, J. Schiller, and J. Seitz. *Verteilte Informationssysteme - Integration von Datenübertragungstechnik und Datenbanktechnik*. dpunkt Verlag, Heidelberg, 2003.

[Alt] M. Altus. A modular design strategy for a flexible graphical database design environment: An experimental study. pages 146–162.

[Alt96] S. Alter. *Information systems: A management perspective*. Benjamin/Cummings, Menlo Park, 2nd edition, 1996.

[BCN92] C. Batini, S. Ceri, and S. Navathe. *Conceptual database design (an entity-relationship approach)*. Benjamin/Cummings, Redwood City, 1992.

[BDK01] E. Best, R. Devillers, and M. Koutny. *Petri net algebra*. Springer, Berlin, 2001.

[Bek92] J. H. Ter Bekke. *Semantic data modelling*. Prentice-Hall, London, 1992.

[Bes95] E. Best. *Semantik: Theorie sequentieller und paralleler Programmierung*. Vieweg, Wiesbaden, 1995.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley, Reading, MA, 1987.

[Bis95] J. Biskup. *Foundations of information systems*. Vieweg, Wiesbaden, 1995. In German.

[BL93] A. J. Bernstein and P. M. Lewis. *Concurrency in programming and database systems*. Jones and Bartlett, Sudbury, MA, 1993.

[BP98] M. Blaha and W. Premerlani, editors. *Object-oriented modeling and design for database applications*. Prentice-Hall, Upper Saddle River, 1998.

[Bra94] M. H. Brackett. *Data sharing: Using a common data architecture*. John Wiley & Sons, 1994.

[Bro00] L. Brown. *Integration models - Templates for business transformation*. SAMS Publishing, New York, 2000.

[BS97] J. Barwise and J. Seligman. *The logic of distributed systems*. Cambridge University Press, Cambridge, 1997.

[BS03] E. Börger and R. Stärk. *Abstract state machines - A method for high-level system design and analysis*. Springer, Berlin, 2003.

[CP84] S. Ceri and G. Pelagatti. *Distributed databases: Principles and systems*. McGraw-Hill, New York, 1984.

[Dad96] P. Dadam. *Verteilte Datenbanken und Client/ServerSysteme*. Springer, Berlin, 1996.

[DGH03] S. Dustdar, H. Gall, and M. Hauswirth. *Software-Architekturen für verteilte Systeme*. Springer, 2003.

[Dre95] H. Dreßler. *Datenstrukturentwurf - Vom Faktenchaos zur Anwenderdatenbank*. Oldenbourg-Verlag, München, 1995.

[DT04] A. Düsterhöft and B. Thalheim. Linguisitc based search facilities in snowflake-like database schemes. *Data & Knowledge Engineering*, 48(1):177–198, 2004.

[Ebe94] R. E. Eberts. *User interface design*. Prentice-Hall, Englewood Cliffs, 1994.

[Elm92] A. K. Elmagarmid, editor. *Database transaction models for advanced applications.* Morgan Kaufmann, San Mateo, 1992.

[Emb98] D. W. Embley. *Object database development: Concepts and principles.* Addison-Wesley, Reading, Mass., 1998.

[EN00] R. Elmasri and S. Navathe. *Fundamentals of database systems.* Addison-Wesley, Reading, 2000.

[Fer95] T. Fernandes. *Global interface design.* Addison-Wesley, Boston, 1995.

[FS95] S. Fowler and V. Stanwick. *The GUI style guide.* Academic Press, Amsterdam, 1995.

[FvH89] C. C. Fleming and B. von Halle. *Handbook of relational database design.* Addison-Wesley, Reading, MA, 1989.

[Gil90] M. L. Gillenson. *Database step-by-step.* John Wiley & Sons, New York, second edition, 1990.

[Glo96] P. Gloor. *Elements of hypermedia design: Techniques for navigation & visualization in cyberspace.* Birkhauser, Boston, 1996.

[GMUW00] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems implementation.* Prentice-Hall, 2000.

[GR94] J. Gray and A. Reuter. *Transaction processing: Concepts and techniques.* Morgan Kaufmann, San Mateo, 1994.

[Gur00] Y. Gurevich. Sequnetial abstract-state machines capture sequential algorithms. *ACM TOCL*, 1(1):77–111, 2000.

[Hal95] T. A. Halpin. *Conceptual schema and relational database design.* Prentice-Hall, Sydney, 1995.

[Hau00] R. Hausser. *Foundations of computational linguistics.* Springer, Berlin, 2000. in German.

[Haw90] I. T. Hawryszkiewycz. *The art of database design.* MacMillan, 1990.

[Hor99] I. Horrocks. *Constructing the user interface with statecharts.* Addison-Wesley, Harlow, 1999.

[HP97] L. J. Heinrich and G. Pomberger. Theorie und Praxis der Wirtschaftinformatik 9, 1997.

[IZG97] W. H. Inmon, J. A. Zachman, and J. G. Geiger. *Data stores, data warehousing and the Zachman framework.* McGraw Hill, New York, 1997.

[Kas03] R. Kaschek. *Konzeptionelle Modellierung.* Habilitationsschrift, Universität Klagenfurt, 2003.

[KE96] A. Kemper and A. Eikler. *Datenbanksysteme.* Oldenbourg-Verlag, München, 1996.

[KE01] A. Kemper and A. Eikler. *Datenbanksysteme.* Oldenbourg Verlag, München, 2001.

[KK93] P. Kandzia and H.-J. Klein. *Theoretische Grundlagen relationaler Datenbanksysteme.* Bibliographisches Institut, Darmstadt, 1993.

[KM03] M. Klettke and H. Meyer. *XML & Datenbanken - Konzepte, Sprachen und Systeme.* dpunkt.verlag, Heidelberg, 2003.

[KT95] E. Küthe and M. Thun. *Marketing mit Bildern: Management mit Trend-Tableaus, Mood-Charts, Storyboards, Fotomontagen und Collagen.* DuMont, Köln, 1995.

[Kun92] J. Kunze. Generating verb fields. In *Proc. KONVENS*, Informatik Aktuell, pages 268–277. Springer, 1992. in German.

[Leo92] M. Leonard. *Database design theory.* MacMillan, Houndsmills, 1992.

[LFe] C. Löckenhoff, D. Fensel, and R. Studer (eds.). *CommonKADS, Proc. 3rd KADS meeting.*

[LL99] M. Levene and G. Loizou. *A guided tour of relational databases and beyond.* Springer, Berlin, 1999.

[LL03] M. Levene and G. Loizou. Why is the snowflake schema a good data warehouse design? *Information Systems*, 28(3):225–240, 2003.

[LM78] P. C. Lockemann and H. C. Mayr. *Computer-based information systems.* Springer, Berlin, 1978. In German.

[Mac90] L. A. Maciaszek. *Database design and implementation.* Prentice Hall, Sydney, 1990.

[Mai83] D. Maier. *The theory of relational databases.* Computer Science Press, Rockville, MD, 1983.

[Mok96] C. Mok. *Designing business.* Hayden, 1996.

[Mor03] T. Moritz. Szenographie interaktiver informationssysteme. BTU Cottbus, Informatik, Manuskript, 2003.

[MR92] H. Mannila and K.-J. Räihä. *The design of relational databases.* Addison-Wesley, Wokingham, England, 1992.

[MS95] K. Mullet and D. Sano. *Designing visual interfaces.* Prentice-Hall, Englewood Cliffs, 1995.

[Pae00] B. Paech. *Aufgabenorientierte Softwareentwicklung.* Springer, Berlin, 2000.

[PBGG89] J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The structure of the relational database model.* Springer, Berlin, 1989.

[Pro72] V.J. Propp. *Morphologie des Märchens.* Carl Hanser Verlag, München, 1972.

[Rah94] E. Rahm. *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung.* Addison-Wesley, Bonn, 1994.

[RG94] M. Reingruber and W. W. Gregory. *The data modeling handbook.* John Wiley & Sons, New York, 1994.

[Rie99] J. Rieckmann. Component ware for supporting transport logistics. In B. Scholz-Reiter, H.-D. Stahlmann, and A. Netke, editors, *Proc. Process Modelling*, pages 256–275. Springer, Berlin, 1999.

[Ris88] N. Rishe. *Database design fundamentals.* Prentice-Hall, Englewood Cliffs, 1988.

[RS97] O. Rauh and E. Stickel. *Konzeptuelle Datenmodellierung.* Teubner, Stuttgart, 1997.

[Sch96] B. Schewe. *Kooperative Softwareentwicklung - Ein objektorientierter Ansatz.* Deutscher Universitäts-Verlag, Wiesbaden, 1996.

[Sim94] G. Simsion. *Data modeling essentials - Analysis, design and innovation.* Van Nonstrand Reinhold, New York, 1994.

[Teo89] T. J. Teorey. *Database modeling and design: The entity-relationship approach.* Morgan Kaufmann, San Mateo, 1989.

[Tha91] B. Thalheim. *Dependencies in relational databases.* Teubner, Leipzig, 1991.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology.* Springer, Berlin, 2000. See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm.

[Tha01] B. Thalheim. Abstraction layers in database structuring: The star, snowflake and hierarchical structuring. Technical Report Preprint I-13-2001, Brandenburg University of Technology at Cottbus, Institute of Computer Science, 2001. See also: http://www.informatik.tu-cottbus.de/∼thalheim/slides.htm.

[Tha02] B. Thalheim. Component construction of database schemes. In *Proc. ER'02*, volume 2503 of *LNCS*, pages 20–34. Springer, Berlin, 2002.

[Tha03] B. Thalheim. Visual sql - an er based introduction into database programming. Technical Report 08/03, Brandenburg University of Technology at Cottbus, Insitute of Computer Science, May 2003 2003.

[Tsa89] M. Sh. Tsalenko. *Modeling of semantics for databases.* Nauka, Moscov, 1989. (in Russian).

[Web95] B. F. Webster. *Pitfalls of object-oriented development: a guide for the wary and entusiastic.* M&T books, New York, 1995.

[YTS+99] S. Yigitbasi, B. Thalheim, K. Seelig, S. Radochla, and R. Jurk. Entwicklung und Bereitstellung einer Forschungs- und Umweltdatenbank für das BTUC Innovationskolleg. In F. Hüttl, D. Klem, and E. Weber, editors, *Rekultivierung von Bergbaufolgelandschaften*, pages 269–282. Walter de Gruyter, Berlin, 1999.

Danksagung für die Mitarbeit

Diese Arbeit stellt eine Zusammenfassung der Arbeiten zum Co-Design-Modell dar. Sie ist in vielen Diskussionen mit den Mitgliedern meiner Arbeitsgruppen in Rostock und Cottbus entstanden, denen ich sehr danke. Wir haben in Cottbus parallel zwei Co-Design-Ansätze entwickelt: den hier behandelten Top-Down-Zugang und einen Bottom-Up-Zugang unter der Leitung von Wolfram Clauß und später Thomas Feyer auf der Basis von Stellen-Transitionsnetzen. Beide Zugänge haben ihre Berechtigung. Methodisch ist der erste Zugang einfacher. Für Detailbereiche, wie z.B. die Interaktionsmodellierung, ist die zweite Methode einfacher und effizienter. Inhaltlich sind beide Zugänge gleichwertig. In der Praxis wird sich ein Mix dieser beiden Methoden durchsetzen.

Eine Methode ist erst dann "reif", wenn sie sowohl in der Entwicklungspraxis als auch in der Lehre ihren Niederschlag findet. In der Lehre wurde diese Methodik aller vier Semester in mehr als einem Dutzend Jahren erprobt. Es haben dabei viele Studenten mit kritischen Hinterfragungen auch zu einer Verfeinerung der Methode beigetragen.

Die Co-Design-Methode ist mit den Mitgliedern der Arbeitsgruppe von Klaus-Dieter Schewe (Massey University, Palmerston North) weiter verfeinert worden. Ihm und Roland Kaschek möchte ich extra danken.

Die Co-Design-Methode hat ihre praktische Erprobung mit ID^2 bereits seit längerer Zeit erfahren. Ich danke meinen Studenten und Kollegen aus Kuwait, Oman und Jordanien für die Unterstützung, die Anregungen und vor allem für die Herausforderungen. Mit der Vielzahl von Anwendungen in Arabien habe ich zu den Anforderungen, zu den Tricks und kleinen Brücken, die jede Methode erfordert, und zu großen Anwendungen sehr viel gelernt.

Das $(DB)^2$-System wurde zum RADD-System durch die Mitglieder meiner Rostocker und Cottbuser Arbeitsgruppen erweitert . Die Entwicklungsarbeiten meiner Studenten und Promoventen, insbesondere von Margita Altus, Antje Düsterhöft und Meike Klettke haben zu einem ausgereiften System geführt, dessen Kernideen in das System ID^2 übernommen wurden.

Die Co-Design-Methode hat auch viele Anwendungen zur Entwicklung von informationsintensiven Websites gefunden. Das Cottbuser InfoTeam hat mehr als 35 große Websites entwickelt. Im wesentlichen wurde die Co-Design-Methode verwendet, was auch gleichzeitig zu Herausforderungen an der Methodik führte.

Die Bewertung der Methodik nach dem SPICE 2.0 Framework wurde durch die Gruppe um H. Jaakkola in Pori dankenswerterweise 2003 durchgeführt. Diese Zusammenschrift ist während meines Sabbaticals im Sommersemester 2003 entstanden. Ich danke insbesondere meinen ungarischen Freunden Gyula Katona und Janos Demetrovics für die jahrelange Unterstützung und für das Refugium in der letzten Phase des Zusammenschreibens.

Ein besonderer Dank gebührt dem "guten Geist des Lehrstuhles", Karla Kersten.

Bitte

Diese Arbeit wird nicht endgültig fertig gestellt sein. Jedem, der kritische Bemerkungen und auch Verbesserungsvorschläge hat, möchte ich bitten, mir diese nicht vorzuenthalten sondern unter

thalheim@is.informatik.uni-kiel.de

zukommen zu lassen.

Weiterentwicklung

Die Co-Design-Methode ist durch eine detaillierte Entwicklungsmethodik unterlegt worden. In einer weiteren Arbeit wird diese Entwicklungsmethodik des schrittweisen Entwurfes vertiefend behandelt werden.

Verzeichnis der Illustrationen

Index